



TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií ■

Softwarové rozhraní pro laserový triangulační senzor TLE1

Diplomová práce

M12000213

Studijní program: N2612 – Elektrotechnika a informatika

Studijní obor: 1802T007 – Informační technologie

Autor práce: **Bc. Tomáš Němeček**

Vedoucí práce: Ing. Miroslav Holada, Ph.D.





TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies ■

Software interface for laser triangulation sensor TLE1

Diploma thesis

M12000213

Study programme: N2612 – Electrical Engineering and Informatics

Study branch: 1802T007 – Information technology

Author: **Bc. Tomáš Němeček**

Supervisor: Ing. Miroslav Holada, Ph.D.



ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Bc. Tomáš Němeček**
Osobní číslo: **M12000213**
Studijní program: **N2612 Elektrotechnika a informatika**
Studijní obor: **Informační technologie**
Název tématu: **Softwarové rozhraní pro laserový triangulační senzor TLE1**
Zadávající katedra: **Ústav informačních technologií a elektroniky**

Z á s a d y p r o v y p r a c o v á n í :


1. Seznamte se s laserovým triangulačním senzorem TLE1.
2. Navrhněte softwarové rozhraní usnadňující komunikaci se senzorem z prostředí C#.
3. Navržené rozhraní realizujte včetně dokumentace.
4. Možnosti snímače a navrženého rozhraní otestujte v ukázkové aplikaci, která bude demonstrovat využitelnost navrženého rozhraní.

Rozsah grafických prací: Dle potřeby dokumentace
Rozsah pracovní zprávy: cca 40 - 50 stran
Forma zpracování diplomové práce: tištěná/elektronická
Seznam odborné literatury:

- [1] PINKER, Jiří: Mikroprocesory a mikropočítače. 1. vyd., 1. dot. Praha: BEN -
- [2] technická literatura, 2008, 159 s. ISBN 978-80-7300-110-0.
- [3] BREY, B., B.: Applying PIC18 Microcontrollers: Architecture, Programming, and Interfacing using C and Assembly. Prentice Hall, 2008, ISBN-10: 0130885460.
- [4] GOOK, M.: Hardwarová rozhraní - Průvodce programátora. Computer press, Praha, 2006, ISBN 80-251-1019-2.
- [5] NAGEL, Christian; EVJEN, Bill, C# 2008: Programujeme profesionálně. 1. vyd.. Computer press, Praha, 2009, 1904 s. ISBN 978-80-251-2401-7.
- [6] SHARP, John: Microsoft Visual C# 2010. 1. vyd., Computer Press, Brno, 2010, 694 s. ISBN 978-80-251-3147-3.

Vedoucí diplomové práce: Ing. Miroslav Holada, Ph.D.
Ústav informačních technologií a elektroniky

Datum zadání diplomové práce: 12. září 2013
Termín odevzdání diplomové práce: 16. května 2014


prof. Ing. Václav Kopecký, CSc.
děkan

L.S.


prof. Ing. Zdeněk Plíva, Ph.D.
vedoucí ústavu

V Liberci dne 12. září 2013

Prohlášení

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu TUL.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé diplomové práce a konzultantem.

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum:

Podpis:

Poděkování

Rád bych poděkoval svému vedoucímu panu Ing. Miroslavu Holadovi, Ph.D. za vedení mé práce a firmě Metralight za zapůjčení senzoru, trpělivost a poskytnutí mnoha cenných rad.

Abstrakt

Cílem diplomové práce je návrh softwarového rozhraní pro laserový triangulační senzor TLE1 firmy Metralight a demonstrace jeho schopností pomocí ukázkové aplikace, která rozhraní implementuje. K řešení byl použit programovací jazyk C# .NET a framework Windows Presentation Foundation. Zadaný problém byl řešen studováním principu práce senzoru a jeho komunikačního protokolu. Následoval návrh softwarového rozhraní, analýza možných problémů při budoucích změnách senzoru a vytvoření metod, pro které bylo zásadní, aby obsáhly všechny funkce senzoru TLE1. Dále bylo rozhraní ověřeno pomocí ukázkové aplikace, která je právě na vytvořeném rozhraní postavena. Rozhraní bylo zároveň otestováno firmou Metralight, která jej nabídla svým zákazníkům. Ti již nejsou nuceni při programování aplikací studovat princip práce komunikačního protokolu senzoru. Výsledky této diplomové práce umožňují významné zrychlení práce se senzorem TLE1 a jeho snadnější implementaci v rámci vlastních systémů.

Klíčová slova

laserový triangulační senzor, softwarové rozhraní, programovací jazyk C#, vizualizace dat, komunikační protokol

Abstract

The aim of this work is to design a software interface for laser triangulation sensor TLE1 from Metralight Company and demonstration of his capabilities with exemplary application which implements the interface. For programming has been used C# .NET programming language with Windows Presentation Foundation framework. Specified problem has been solved by studying work principle of the sensor and its communication protocol, followed by proposal of software interface and analysis of possible problems in future revisions of the sensor and the development of methods for which it was essential to contain all the functionality of the sensor TLE1. It was also necessary to verify the interface using a exemplary application that will implement the interface. The interface was also tested in the Metralight Company which offered the interface to their customers, who are no longer forced to study the working principle of the sensor communication protocol. Results of this thesis allows significant simplification of using TLE1 sensor and easier way for implementing this sensor within own systems.

Key words

laser triangulation sensor, interface, C# programming language, data visualization, communication protocol

Obsah

Úvod	12
2 Senzor TLE1	13
2.1 Parametry senzoru	13
2.2 Uživatelské parametry	15
2.3 Komunikační protokol	16
3 Softwarové rozhraní	23
3.1 Jmenná konvence	24
3.2 Komunikační rozhraní	24
3.3 Implementace funkcí senzoru	26
3.4 Třída SensorClass	31
3.5 Výsledná struktura rozhraní	36
4 Realizace ukázkové aplikace	38
4.1 TLE Studio	38
4.2 Profile Scanner	48
5 Závěr	52
Literatura	53
Přílohy	54
A Rozměry senzoru TLE1	54
B EEPROM mapa	55
C Volatilní parametry senzoru	56
D Obsah vloženého CD	57
E Dokumentace k rozhraní TLESensorClass	58

Seznam obrázků

2.1	Senzor TLE1	13
2.2	Rozměry měřené senzorem	14
2.3	Obrazová data a k nim odpovídající data profilová	19
3.1	UML diagram tříd pro připojení	25
4.1	Okno aplikace TLE Studio	40
4.2	Záložka Sensor aplikace TLE Studio	42
4.3	Záložka X Distance aplikace TLE Studio	43
4.4	Záložka Custom aplikace TLE Studio	44
4.5	Záložka Image aplikace TLE Studio	46
4.6	Záložka Profile aplikace TLE Studio	47
4.7	Záložka EEPROM aplikace TLE Studio	48
4.8	Aplikace Profile Scanner	49
4.9	Nasnímané objekty programem Profile Scanner	51

Seznam tabulek

2.1	Položky AUX bajtu	17
2.2	Volatilní parametry senzoru	21

Seznam zdrojových kódů

3.1	Abstraktní třída pro komunikační rozhraní	25
3.2	Komunikační rozhraní	27
3.3	Zpracování dodatečných informací	28
3.4	Zpracování EEPROM dat	29
3.5	Odemčení senzoru	32
3.6	Odeslání příkazu senzoru	32
3.7	Čtení měřených dat	33

3.8	Šablona funkce pro čtení parametrů	35
3.9	Čtení parametru	35
3.10	Šablona funkce pro uložení parametru	36

Seznam zkratek

BMP	Bitmap, Formát pro ukládání rastrové grafiky
EEPROM	Electrically Erasable Programmable Read-Only Memory, Paměť umožňující mazání elektronickým pulzem
MSB	Most Significant Bit, Nejvyšší bit
POE	Power Over Ethernet, Způsob přenosu napětí přes ethernetové rozhraní
ROI	Region of Interest, Oblast v rámci které senzor zpracovává data
USB	Universal Serial Bus, Univerzální sériové komunikační rozhraní
WPF	Windows Presentation Foundation, Framework jazyka C# .NET umožňující tvorbu grafických komponent

Úvod

Diplomová práce se zabývá problematikou návrhu softwarového rozhraní pro laserový senzor TLE1. Cílem práce je vytvořit prostředek pro jednodušší práci se senzorem pro uživatele bez pokročilých znalostí jeho principu měření či komunikačního protokolu a demonstrovat jeho použití vytvořením ukázkové aplikace, která jej implementuje a zároveň naznačí možná využití senzoru v praxi. Senzor TLE1 byl vybrán na základě předchozí spolupráce se společností Metralight, která senzor vyrábí a která chce svým zákazníkům nabídnout vhodné řešení pro urychlení zařazení senzoru do jejich systémů či výrobních linek.

Diplomová práce stručně shrnuje vlastnosti senzoru TLE1. Je zde popsán princip měření, jeho komunikační protokol, nastavitelné parametry a práce s pamětí. Dále se zabývá samotným procesem vývoje softwarového rozhraní od jeho návrhu, přes implementaci všech funkcí a parametrů, zpracování několika možných typů měřených dat a vyřešení problému budoucího přidání dalších komunikačních rozhraní. Dále je představeno použití vytvořeného rozhraní v rámci dvou ukázkových aplikací. První má za úkol prověřit funkčnost navrženého rozhraní, demonstraci všech vlastností senzorů TLE1 a vhodnou vizualizaci měřených a obrazových dat. Druhá aplikace má naznačit možné způsoby využití senzoru jak v rámci Technické univerzity v Liberci, tak v praxi.

2 Senzor TLE1

TLE1 je laserový triangulační senzor vyrobený společností Metralight sloužící k velmi přesnému měření vzdálenosti od objektu a jeho výšky. Na rozdíl od podobných senzorů nepracuje s úzkým svazkem světla a měřením v jediném bodě, ale na objekt vyše širší svazek, který následně zachytí pomocí CMOS čidla s rozlišením 1280×1024 pixelů. Senzor je díky většímu množství bodů schopen přesněji měřit například rozměry otvorů v objektech nebo zachytit profil výrobku.



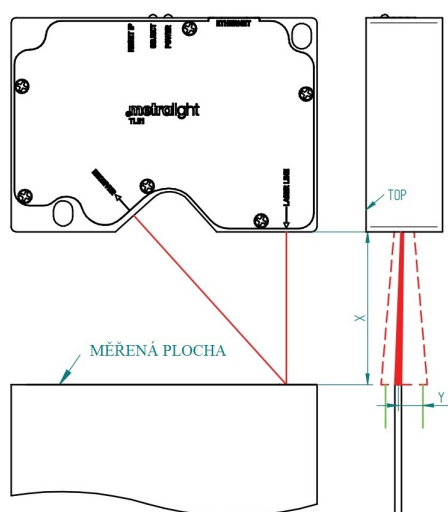
Obrázek 2.1: Senzor TLE1

2.1 Parametry senzoru

Jak již bylo řečeno výše, jedná se o senzor pro měření vzdálenosti a výšky pracující na principu laserové triangulace. Rozlišení tohoto měření dosahuje $1 \mu\text{m}$ pro měření vzdálenosti a méně než $20 \mu\text{m}$ pro měření výšky. Oblast měření začíná 35 mm od hrany senzoru a končí na 65 mm. S okolím senzor komunikuje pomocí ethernetového

rozhraní a je možné využít jak TCP, tak UDP protokol. Přes ethernetové rozhraní je senzor současně i napájen použitím funkce POE. Využití POE ale zároveň znamená omezení maximální komunikační rychlosti na 100 Mbit/s. Rozměry senzoru jsou uvedeny v příloze A.

Senzor pracuje ve čtyřech základních režimech měření. V režimu MODE_0 měří pouze vzdálenost (osa X) od objektu průměrováním hodnot uživatelem definovaného množství bodů. V režimu MODE_1 měří kromě vzdálenosti i polohu objektu v ose Y. V režimech MODE_2 a MODE_3 senzor měří vzdálenost od objektu v ose X a v ose Y určuje pozici objektu od pravé resp. levé strany. Manuál [Metralight \(2012\)](#) označuje druhý měřený rozměr jako „height” – tedy výška. Tento pojem je však zavádějící, protože se nejedná o výšku prostorovou, ale výšku ve 2D. Konkrétní ukázkou měřených rozměrů při aktivovaném režimu MODE_1 ukazuje obrázek 2.2. V textu dále bude ale pro zachování jednotnosti pojmů tento rozměr označován jako výška.



Obrázek 2.2: Rozměry měřené senzorem

2.2 Uživatelské parametry

Uživatel má možnost ovlivnit práci senzoru nastavením několika parametrů. Parametry se dělí na nevolatilní, které jsou uloženy ve vnitřní EEPROM paměti, a volatilní, které se při každém restartování senzoru nastaví na hodnoty, které jsou uloženy v EEPROM paměti. Změna volatilních parametrů bude mít tedy vliv na práci senzoru jen do doby, dokud je senzor zapnut.

2.2.1 Nevolatilní parametry

Nevolatilní uživatelské parametry jsou v senzoru uloženy na devíti stránkách paměti EEPROM. Na stránce s adresou 255 jsou uloženy následující parametry nastavující ethernetové připojení:

- **IP adresa senzoru**
- **Maska podsítě**
- **Výchozí brána**

Na stránkách 247 až 254 jsou uloženy ostatní parametry, přičemž na každé stránce jsou uloženy ty samé parametry ve stejném pořadí, což umožňuje vytvořit 8 přednastavení, která lze během práce se senzorem přepínat. Parametry jsou uloženy v následujícím pořadí.

- **Peak Size From** – Minimální hodnota signálu, která bude zpracována
- **Peak Size To** – Maximální hodnota signálu, která bude zpracována
- **Line Processing Mode** – Režim zpracování měřených hodnot
- **Default Mode** – Režim měření, který senzor po startu nastaví
- **Offset X** – Přičtení / odečtení hodnoty k měřené hodnotě v ose X
- **Offset Y** – Přičtení / odečtení hodnoty k měřené hodnotě v ose Y

- **Laser Power** – Výkon laserového vysílače
- **Time Of Integration** – Doba zpracování paprsku na CMOS snímači
- **Threshold** – Hodnota prahu, pod kterým se již data nezpracovávají
- **Average Window** – Nastavení okénkovací funkce
- **Region of Interest** – Nastavení oblasti, v rámci které se zpracovávají data
- **Window** – Nastavení oblasti, v rámci které senzor měří

Hodnoty, které jsou větší než 256, jsou rozděleny na dva bajty a nižší bajt je uložen před vyšším. Kompletní přehled všech paramterů (i těch uživatelsky nepřístupných), jejich adres a stránek paměti je zobrazen v příloze B.

2.2.2 Volatilní parametry

Volatilní parametry odpovídají z větší části parametrům nevolatilním s tím rozdílem, že mezi volatilní parametry nepatří ethernetové parametry a parametry jako Default Mode a Offset, které se nastavují po startu senzoru.

2.3 Komunikační protokol

Komunikační protokol senzoru TLE1 je založen na bajtových příkazech. Příkazy pro senzor mohou být jak jednobajtové, tak i vícebajtové. Současně i odpověď od senzoru může mít různou délku.

2.3.1 Měřená data

Žádost o měřená data, tedy informace o vzdálenosti a výšce, má formát jednoho bajtu s hodnotou $\langle 0x1X \rangle$, kde X znamená množství požadovaných dat. Senzor odpoví 2^x měřenými daty. Pokud jsou tedy požadována jen jedna měřená data, bude příkaz pro senzor 0x10. Každá odpověď měřených dat má velikost 5 B a jejich formát je následující:

Přijatá data: <X VYŠŠÍ BAJT> <X NIŽŠÍ BAJT><Y VYŠŠÍ BAJT>
<Y NIŽŠÍ BAJT> <AUX BAJT>

Rozsah měřených dat X a Y je od 0 do 30000. Číslo představuje vzdálenost resp. výšku v mikrometrech. Formát posledního (AUX) bajtu je uveden v tabulce 2.1.

Tabulka 2.1: Položky AUX bajtu

bit	Název	Popis
7 (MSB)	OIN	Objekt detekován v měřícím rozsahu
6	ZERO_CNT	Objekt je uvnitř procesovacího okna
5	OVER410_CNT	Signalizuje, že objekt překročil hodnotu 410 px uvnitř procesovacího okna
4	0	Rezervováno
3	USER_PAR_CHG	Uživatelské parametry se po zapnutí změnily
2...0	MODES	Tříbitové číslo reprezentující nastavený režim měření (viz. 2.1)

2.3.2 Obrazová data

Senzor umožňuje vyčtení celého obrazu zachycovaného pomocí CMOS čidla. Rozlišení tohoto obrazu je závislé na konkrétní nastavené hodnotě parametru Window – tedy rozsahu, ve kterém senzor měří. Zároveň je možné vyčíst obraz se čtvrtinovým rozlišením. Žádost o obrazová data má opět podobu jednoho bajtu, tentokrát s hodnotou <0xD0> pro plnou velikost a <0xD1> pro čtvrtinovou velikost.

Odpověď senzoru má podobu bajtového pole o velikosti rozlišení nastaveného parametru Window. Položky pole představují jednotlivé pixely šedotónového obrazu.

Obraz je zasílán po řádcích bez ukončujícího znaku – programátor si tak sám musí udržovat informaci o rozlišení.

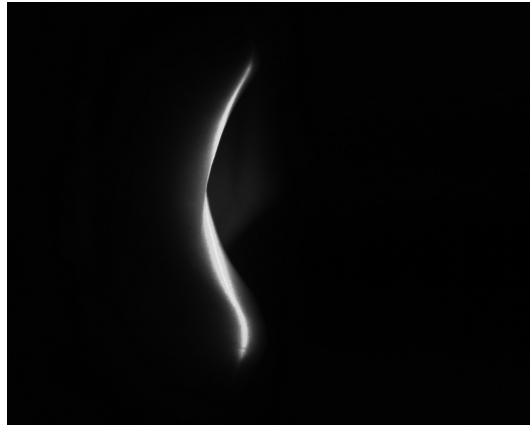
S obrazovými daty souvisí i vyčtení tzv. profilu. Tedy zpracovaných obrazových dat ve formě dvourozměrného pole se vzdáleností na ose X a výškou na ose Y. Velikost profilu je opět závislá na nastavení parametru Window. Pro přečtení těchto dat je nutné poslat příkaz <0x60>. Senzor odpoví zasláním bajtového pole, které má následující formát:

Přijatá data: <X VYŠŠÍ BAJT 1> <X NIŽŠÍ BAJT 1> <Y VYŠŠÍ BAJT 1>
<Y NIŽŠÍ BAJT 1> ... <X VYŠŠÍ BAJT N> <X NIŽŠÍ BAJT N>
<Y VYŠŠÍ BAJT N> <Y NIŽŠÍ BAJT N> <POŘADÍ VYŠŠÍ BAJT>
<POŘADÍ NIŽŠÍ BAJT>

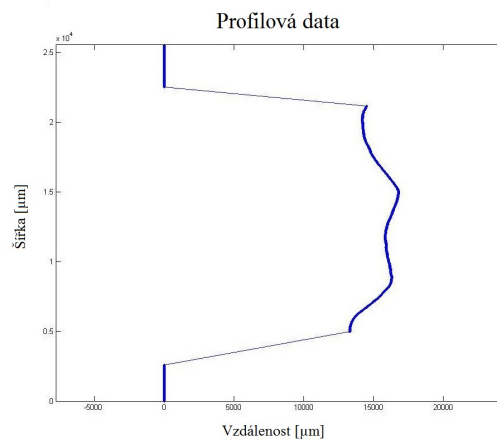
Počet bodů odpovídá nastavené výšce procesovacího okna Window. Na konci pole jsou ještě odeslány dva bajty obsahující pořadové číslo profilu. To je důležité zejména v režimu nepřetržitého příjmu profilových dat k identifikaci konkrétního profilu. Celková velikost pole tedy je:

$$\text{velikost} = (4 \times \text{výška procesovacího okna}) + 2$$

Na obrázku 2.3a je zobrazen příklad obrazových dat a k nim na obrázku 2.3b odpovídajících dat profilových. Profilová data jsou převrácena kolem osy Y, protože jsou tak senzorem zpracovávána.



(a) Obrazová data



(b) Profilová data

Obrázek 2.3: Obrazová data a k nim odpovídající data profilová

2.3.3 Stream dat

Kromě běžného modelu žádost – odpověď umožňuje senzor i stream dat, tedy nepřetržitý proud informací od vyslání příkazu k zahájení po odeslání příkazu pro ukončení. Výhoda streamu je hlavně v tom, že senzor data posílá maximální možnou rychlostí a žádná data nechybí. Pokud by byl například měřen rychle se pohybující objekt, mohlo by u běžného modelu žádost – odpověď dojít k tomu, že v datech, která budou chybět, bude důležitá informace. Vyřízení žádosti senzoru chvíli trvá, zatímco u streamu dat nemusí senzor žádost pokaždé zpracovávat.

Senzor umožňuje streamovat dva typy dat – měřená data a profilová data. Pro zahájení streamu měřených slouží příkaz **<0x21>**, pro stream profilových dat pak příkaz **<0x22>**. Senzor po přijetí tohoto příkazu začne okamžitě odesílat data. Pro ukončení streamu je potřeba odeslat příkaz **<0x20>**. Na příkaz pro ukončení streamu senzor nijak neodpoví.

2.3.4 Práce s EEPROM pamětí

Data v EEPROM paměti jsou uložena v osmi různých stránkách, přičemž v každé jsou data uložena ve stejném pořadí (viz. 2.2.1). Při zápisu do této paměti je možné pracovat pouze s jednou stránkou zároveň. Senzor by uměl zpracovat i více stránek zároveň, uživateli však tato možnost není z bezpečnostních důvodů umožněna. Příkaz pro uložení dat do EEPROM paměti má následující strukturu:

Odesílaná data: **<0xB0>** **<ČÍSLO STRÁNKY>** **<OFFSET>**
<POČET BAJTŮ – 1> **<BAJT 1>** **<BAJT 2>** ... **<BAJT N>**

Senzor umožňuje najednou zapsat až 256 bajtů dat do stránek od 247 do 255. Data nejsou senzorem nijak ověřována. Je tak na programátorovi, aby zajistil, že do senzoru ukládá správná data. Senzor potvrdí provedení příkazu odesláním bajtu **<POČET BAJTŮ – 1>** z odeslané zprávy zpět uživateli.

Obdobný příkaz slouží i k přečtení údajů na jedné stránce EEPROM paměti. Má následující strukturu:

Odesílaná data: **<0xA0>** **<ČÍSLO STRÁNKY>** **<OFFSET>**
<POČET BAJTŮ – 1>

Na tento příkaz senzor odpoví odesláním bajtového pole o jeden bajt většího než uživatel definoval ve čtvrtém bajtu odeslaného příkazu.

2.3.5 Nastavení volatilních parametrů senzoru

Nastavení volatilních parametrů má pokaždé stejnou strukturu. Při čtení uživatel odešle jeden bajt a jako odpověď dostane bajty dva (vyšší bajt první). Při zápisu uživatel odešle jeden bajt specifikující parametr a pak samotnou hodnotu ve dvou bajtech (vyšší bajt první). Senzor na příkaz pro zápis v případě úspěšného provedení operace odpoví vrácením bajtu identifikujícím parametr. Tabulka 2.2. uvádí příklad několika parametrů včetně příkazů pro jejich nastavení a přečtení. Ostatní parametry jsou kvůli většímu množství přesunuty do přílohy C. Parametry jsou pojmenovány tak, jak jsou uvedeny v manuálu [Metralight \(2012\)](#).

Tabulka 2.2: Volatilní parametry senzoru

Parametr	Popis	Čtení	Zápis	Rozsah
WINDOW	Průměrovací okno	0x82	8x8A	1 – 1023
THR	Prah průměrování	0x83	0x8B	1 – 254
TINT	Doba integrace	0x84	0x8C	1 – 1024
LSRPWR	Výkon laseru	0x85	0x8D	0 – 255

2.3.6 Ostatní příkazy senzoru

Kromě výše zmíněných příkazů existují i příkazy, které například umí vyčíst verzi firmwaru senzoru, vypnout laser nebo změnit aktivní režim měření. Příkaz pro vyčtení aktuální verze firmwaru senzoru je `<0xF0>`. Senzor odpoví dvěma bajty. Příkaz nastavující režim měření je `<0x3X>`, kde X je rovno módu, který chce uživatel nastavit. Standardní počet módů v senzoru je roven čtyřem, ale upravené verze senzoru mohou mít módů až osm. Senzor odpoví echo příkazem, kdy po úspěšném nastavení režimu vrátí doručenou zprávu zpět. Pokud je z nějakého důvodu nutné vypnout

laser, stačí odeslat příkaz <0x90> a pro opětovné zapnutí <0x91>. Senzor opět odpoví echo příkazem.

2.3.7 Možné problémy při vytváření rozhraní

Komplikací pro vytváření rozhraní je hlavně možnost zákazníka do jisté míry ovlivnit chování senzoru. Zákazník má možnost zažádat o přidání určitých režimů měření dat či vlastních omezení parametrů. Současně má možnost požádat o přidání dalších příkazů do komunikačního protokolu senzoru. Je tak nutné v rozhraní zohlednit možnost budoucích úprav a rozšíření.

Firma Metralight zároveň pracuje na výrobě senzoru s USB rozhraním. Softwarové rozhraní je tak nutné v tomto ohledu udělat dostatečně obecné, aby bylo možné v budoucnu přidávat další komunikační rozhraní bez komplikovanějších zásahů do kódu.

3 Softwarové rozhraní

Jak bylo již naznačeno v poslední části předchozí kapitoly, rozhraní pro senzor TLE1 bude muset vyřešit několik překážek. Rozhraní musí být dostatečně flexibilní na to, aby pokrylo případné změny vlastností senzoru v budoucnu – ať už jde o změny komunikačního protokolu, režimů měření, vlastností parametrů či komunikačního rozhraní senzoru.

Druhým problémem, který je nutné vyřešit, je přiblížit senzor i programátorům, kteří nemají dlouholeté zkušenosti s programováním a nebyli by tak bez rozhraní schopni aplikaci napsat. Firma Metralight nemá ani zájem na tom, aby zákazníci znali dopodrobna komunikační rozhraní senzoru. Vlastnosti senzoru se totiž mohou v čase měnit a bylo by pro firmu náročné, aby všechny zákazníky při každé změně informovala. V případě použitého rozhraní je zákazníkům buď odeslána nová verze knihovny, nebo jsou změny vyřešeny v rámci rozhraní tak, že zákazník nemusí ve své aplikaci nic změnit.

Rozhraní bude využíváno i v rámci společnosti Metralight pro jednodušší vytváření vlastních aplikací, které mohou například demonstrovat využití senzoru pro potřeby zákazníka.

Při vytváření softwarového rozhraní byly k dispozici pouze skripty pro Matlab. Ty do značné míry naznačily cestu, jakou by mělo být rozhraní vytvářeno, ale protože se programovací techniky jazyka C# a Matlabu značně liší, sloužily tyto skripty hlavně jako kontrola správnosti vytvářených funkcí. V rámci společnosti Metralight se pak pracuje i v jazyce VB .NET. Výhoda rozhraní vytvořeném v jazyce C# je ta, že je jej možné využít právě v jazyce VB .NET nebo načíst v rámci prostředí Matlab. Rozhraní se tak stane daleko univerzálnější a přístupnější pro větší množství programátorů.

3.1 Jmenná konvence

V rámci všech zdrojových kódů je použita stejná jmenná konvence, která vychází z knihy [Sharp \(2010\)](#). Pro psaní víceslovných názvů je s výjimkou konstant použit styl psaní zvaný CamelCase – tedy že každé slovo začíná okamžitě za druhým, ale s velkým písmenem. Názvy jmenných prostorů, tříd, enumerátorů a souborů začínají vždy velkým písmenem. Názvy globálních proměnných mají prefix „m_”. Pokud funkce obsahuje parametry, ty mají prefix „_”. Názvy lokálních proměnných a funkcí pak vždy začínají malým písmenem.

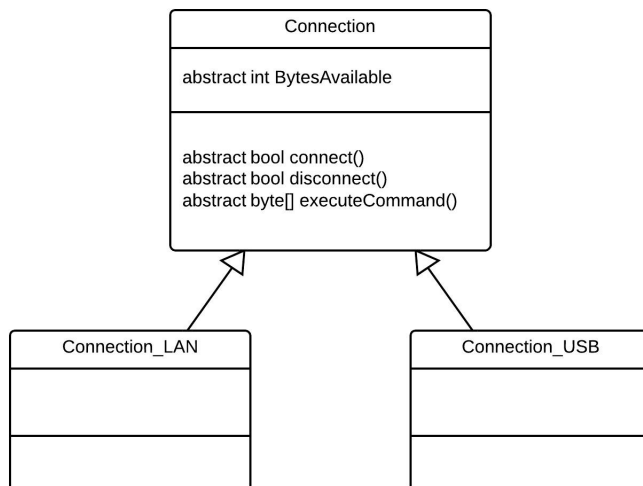
3.2 Komunikační rozhraní

V současné době je senzor vybaven pouze ethernetovým rozhraním. Do budoucna se však počítá s další verzí, která bude vybavena také USB rozhraním. Je proto nutné přizpůsobit rozhraní tak, aby bylo dostatečně obecné a aby přidáním nového způsobu připojení nemusela být měněna celá struktura rozhraní.

Pro tyto účely je možné využít programátorského prostředku – rozhraní, jak je popisováno v [Gamma et al. \(1994\)](#), v něm definovat potřebné metody jako například metodu pro připojení a odpojení a ve třídách definujících jednotlivá rozhraní toto rozhraní implementovat. Nevýhoda rozhraní je ale v tom, že není možné definovat tělo metod definovaných právě uvnitř rozhraní. To je nevýhodou zejména v případě, kdy by bylo vhodné sjednotit metody vypisující řetězce či metody ošetřující chyby. Jazyk C#, podobně jako například jazyk C++ (dle [Prata \(2004\)](#)), pro tyto účely nabízí abstraktní třídy. Ty mají mnoho společných vlastností právě s rozhraním. Pokud se metody uvnitř abstraktní třídy definují jako abstraktní, je programátor, stejně jako v případě rozhraní, nucen metody implementovat případně předefinovat. Výhodou abstraktní třídy je ale právě to, že je možné definovat těla metod.

Protože ale mohou třídy v jazyce C# implementovat rozhraní a zároveň dědit od abstraktních tříd, bylo by možné použít kombinaci obojího. Při využití abstraktních

metod abstraktních tříd ale není nutné rozhraní vůbec použít. Na obrázku 3.1 je naznačena koncepce abstraktní třídy pro připojení k senzoru a zdrojový kód 3.1 pak ukazuje konkrétní použití.



Obrázek 3.1: UML diagram tříd pro připojení

Zdrojový kód 3.1: Abstraktní třída pro komunikační rozhraní

```

1  using System;
3  namespace TLESensorClass.Connections
4  {
5      public enum Status : int { Online, Occupied, Offline };
7      public abstract class Connection
8      {
9          public event EventHandler TimeoutError;
11         public abstract int BytesAvailable { get; }
13         protected void OnTimeoutError(EventArgs e)
14         {
15             EventHandler handler = TimeoutError;
16             if (handler != null)
17             {
18                 handler(this, e);
19             }
20         }
21     }

```

```

21         public abstract Status testConnection();
23         public abstract bool connect();
25         public abstract bool disconnect();
27         public abstract byte[] executeCommand(byte _command,
29             byte[] _data, int _respondSize);
    }
}

```

Zdrojový kód 3.1 zobrazuje abstraktní třídu **Connection**, která definuje abstraktní metody `connect()`, `disconnect()`, `executeCommand()` a `testConnection()`. Tyto metody musí každá třída, která od třídy `Connection` dědí, překrýt. Tedy použít klíčové slovo `override` a díky němu definovat vlastní implementaci zmíněných funkcí.

Význam metod `connect()` a `disconnect()` je zřejmý. Metoda `testConnection()` slouží k testu připravenosti nebo dostupnosti senzoru. Jako výsledek vrátí jeden z prvků enumerátoru `Status`. Metoda `executeCommand()` pak slouží k provedení příkazu pro senzor. Jako parametr přijímá příkaz a případně i jeho data. Zároveň je zde uvedeno, kolik bajtů se má ze senzoru přečíst a tato jsou následně vrácena v podobě bajtového pole. Událost `TimeoutError` definuje akci, která se má stát v případě nedouručení dat v zadaném čase. V tomto případě handler pouze předá událost dál.

3.3 Implementace funkcí senzoru

Aby měl uživatel možnost se senzorem pracovat, bylo nutné implementovat všechny funkce, které senzor poskytuje – tedy přístup k měřeným datům, obrazovým datům, parametrům a funkcím nastavujícím chování senzoru.

3.3.1 Komunikační protokol

Pro usnadnění práce s komunikačním protokolem a zlepšení přehlednosti celého kódu byly všechny příkazy zapsány ve formě enumerátoru. Enumátor je bajtový, takže přímo nahrazuje příkazy. Konkrétní hodnoty příkazů je možné zapsat jak dekadicky, tak šestnáctkově. Konkrétní příklad převodu komunikačního rozhraní je uveden ve zdrojovém kódu 3.2. S jednotlivými příkazy v kódu se tak již nadále nepracuje jako s čísly, ale pomocí jejich identifikátoru.

Zdrojový kód 3.2: Komunikační rozhraní

```
using System;
2
namespace TLESensorClass.Sensor
4 {
    public enum CommandProtocol : byte
6     {
        Get_Profile = 0x60,
8        Get_StartLine_limits = 0x80,
        Get_NumOfLines_limits = 0x81,
10       Get_AverageWindow = 0x82,
        Get_Threshold = 0x83,
12       Get_TimeOfIntegration = 0x84,
        Get_LaserPower = 0x85,
14       Get_LineProcMode = 0x86,
        Get_PeakSize = 0x87
16    };
}
```

3.3.2 Zpracování měřených dat

Přístup ke konkrétním měřeným datům je bez odpovídajícího rozhraní poměrně komplikovaný. Pro získání konkrétních čísel je nutné znát informace o tom, jak jsou řazeny bajty a jaké údaje konkrétně obsahují. Jak bylo popsáno v kapitole 2.3.1, senzor na žádost o měřená data odpoví právě pěti bajty. První dva bajty jsou hodnota vzdálenosti, druhé dva bajty hodnota výšky a poslední bajt je vyhrazen pro

dodatečné informace. Z dodatečných informací se dále v programu používají údaje o tom, zda je objekt v měřené oblasti (současně indikováno diodou přímo na těle senzoru) a o aktuálně aktivním módu měření. Pro účely převodu bajtového pole na konkrétní informace byla vytvořena třída **SensorDataFormat**. Tato třída přijímá jako parametr konstruktoru právě bajtové pole a nabízí pak přístup ke konkrétním hodnotám přes veřejné položky třídy. Pro převod informací z posledního bajtu na dodatečné informace je využita třída **BitArray**, která poskytuje funkce pro přístup k jednotlivým bitům zadaného bajtu. Příklad převodu informace o detekovaném objektu a módu je ukázán ve zdrojovém kódu 3.3.

Zdrojový kód 3.3: Zpracování dodatečných informací

```
1 public SensorDataFormat(byte[] _data)
2 {
3     byte auxByte = _data[4];
4
5     if (auxByte > 0)
6     {
7         BitArray bitArray = new BitArray(auxByte);
8
9         mode = (byte)(auxByte & 0x07);
10
11         try { oin = bitArray.Get(7); }
12         catch (ArgumentOutOfRangeException) { oin = false; }
13     }
14 }
```

Výsledná třída kromě informací uvedených ve zdrojovém kódu 3.3 obsahuje i funkce pro získání ostatních údajů a jsou zde ošetřeny a předány dál všechny vzniklé výjimky a chyby.

Všechny měřené hodnoty je zároveň možné ukládat do historie pro pozdější zpracování určitého množství po sobě jdoucích hodnot. Ukládání hodnot má na starosti třída **ValueHistory**. Ta vytvoří frontu o velikosti, kterou zadá uživatel při jejím vytváření. Protože je však nutné, aby třída zpracovávala údaje velmi rychle (řádově stovky údajů za sekundu), je potřeba použít vhodnou datovou strukturu. Jazyk C#

pro tyto účely poskytuje třídu **Queue**. Ta obsahuje metody `Enqueue()` a `Dequeue()`. Rychlost ukládání a mazání údajů je násobně vyšší než při použití klasického pole nebo Listu. Současně se fronta sama stará o to, aby udržovala pouze určité množství hodnot.

3.3.3 Práce s EEPROM pamětí

V paměti EEPROM je uloženo velké množství údajů. Bylo proto potřeba vytvořit třídu, která tyto údaje zpřehlední. Jak bylo popsáno v kapitole 2.2.1, jsou v EEPROM paměti uloženy údaje dvojího typu. Jsou zde jak parametry senzoru, tak parametry nastavující ethernetové připojení. Z tohoto důvodu byly vytvořeny třídy **EEPROMParameters** a **EEPROMEthernet**. Tyto dvě třídy obsahují metody pro převod bajtového pole získaného ze senzoru na jednotlivé informace a zpět – tedy z konkrétních údajů na bajtové pole. Tyto metody je však možné použít jen při vyčítání kompletní sady parametrů, kdy je přečtena celá stránka paměti. Zdrojový kód 3.4 uvádí jeden z konstruktorů třídy **EEPROMEthernet**, který přijímá jednotlivé parametry a vytváří z nich bajtové pole. Pro urychlení tohoto procesu je použita třída **MemoryStream**. Ta předchází nutnosti vytvářet nové bajtové pole o předem dané velikosti.

Zdrojový kód 3.4: Zpracování EEPROM dat

```
public EEPROMEthernet(IPAddress _address , IPAddress
    _subnetMask , IPAddress _gateway)
2 {
    m_IPaddress = _address;
    4    m_subnetMask = _subnetMask;
    m_gateway = _gateway;
    6
    MemoryStream ms = new MemoryStream(new byte[DATA_SIZE] ,
        0 , DATA_SIZE , true , true);
    8    ms.Write(_address.GetAddressBytes() , 0 , 4);
    ms.Write(_subnetMask.GetAddressBytes() , 0 , 4);
    10    ms.Write(_gateway.GetAddressBytes() , 0 , 4);
    12    m_eepromEthernet_field = ms.GetBuffer();
}
```

3.3.4 Zpracování profilových dat

Jak bylo popsáno v kapitole 2.3.2, profilová data představují předzpracovaný obraz ve formě dvourozměrného pole. Aby bylo možné převést strukturu, kdy jsou jednotlivé body posílány za sebou, do podoby dvou polí nebo jednoho pole dvourozměrného, bylo potřeba vytvořit speciální třídu. Ta se jmenuje **ProfileData** a v konstruktoru přijímá bajtové pole dané velikosti. Toto pole je následně rozděleno na dvě pole, přičemž v jednom poli jsou hodnoty vzdálenosti (X) a ve druhém hodnoty výšky (Y). Pokud bude v budoucnu potřeba rozšířit tuto strukturu o dvourozměrné pole nebo jenom jinou definici jednorozměrného pole, stačí pouze aktualizovat právě tuto třídu.

3.3.5 Práce s parametry

Aby byla sjednocena i práce s parametry, byla vytvořena třída **ActualSettings**. Tato třída poskytuje přístup k proměnným, které se mají v budoucnu použít, anebo se používají za krátkou dobu vícekrát. Jedná se o jednoduchou strukturu složenou z proměnných, které mají veřejné metody get a set. Díky tomu je možné do instance této třídy uložit hodnotu a tu později číst.

Možnost jednotného vyčtení všech parametrů sice byla také diskutována, ale parametrů je tolik, že by to zabralo delší dobu a málokdy je potřeba parametry využít opravdu všechny. Pokud by však taková situace v budoucnu nastala, jednoduchou modifikací třídy **ActualSettings** lze dosáhnout i tohoto chování.

Zvláštním případem parametrů jsou parametry ROI a Window. Tyto parametry se totiž skládají z více dílčích parametrů. Oba mají počáteční řádek, počet řádků, počáteční sloupec a počet sloupců. Jedná se tak v podstatě o obdélníky se zadaným levým horním rohem, šířkou a výškou. Aby nebylo nutné pokaždé přemýšlet, který parametr je vlastně potřeba a jaký parametr se má načíst, byla vytvořena pomocná třída **PointSize**. Tato třída v sobě zapouzdřuje všechny vlastnosti obdélníku a pracuje se všemi parametry zároveň. Jako konkrétní příklad je možné uvést situaci,

kdy programátor potřebuje znát šířku okna Window – buď může použít parametr WINDOW_NUM_OF_COLUMNS anebo intuitivně použít vlastnost window.Height instance třídy PointSize.

3.4 Třída SensorClass

Třída SensorClass v sobě zapouzdřuje všechnu funkcionalitu zmíněnou v předchozích odstavcích a je to právě tato třída, se kterou uživatel pracuje. Po vytvoření instance umožňuje třída se k senzoru připojit, číst a zapisovat do něj data.

Na následujících řádcích budou popsány důležité položky a funkce třídy SensorClass spolu s jejich stručným popisem.

`public SensorClass(Connection _connection)`

Konstruktor třídy, který přijímá jako parametr instanci třídy Connection – ta definuje parametry připojení k senzoru a uloží ji jako neveřejnou proměnnou třídy SensorClass.

`public bool connect()`

Připojí se k senzoru. V případě, že se připojení povede, vrací true. V opačném případě vrací false. K otestování dostupnosti senzoru lze využít funkci testConnection() poskytovanou třídou Connection.

`public bool disconnect()`

Odpojí se od senzoru. Instance řády Connection ale zůstane ve třídě uložena a je možné se k senzoru později zase připojit bez nutnosti znovu zadávat údaje pro připojení.

`public byte[] sensorUnlock()`

Senzoru je odeslána speciální sekvence příkazů, které umožňují práci s EEPROM pamětí či vyčítání obrazu. Odemčení je jakousi bezpečnostní pojistkou pro situace, kdy v programu nastane chyba a program by začal senzoru posílat nesmyslná

data – nedojde tak k náhodnému přepsání paměti EEPROM. V případě úspěšného odemčení odešle senzor zpět první zaslaný bajt. Funkce je vypsána ve zdrojovém kódu 3.6. Funkce `public byte[] sensorUnlock2()` funguje obdobně jako funkce předchozí a liší se pouze v hodnotách příkazů. Bez zadání tohoto příkazu není možné smazat obsah celé paměti EEPROM naráz – musí se mazat po jednotlivých stránkách.

Zdrojový kód 3.5: Odemčení senzoru

```
1 public byte[] sensorUnlock()  
2 {  
3     byte[] recievedData = new byte[1];  
4     recievedData = m_connection.ExecuteCommand(0, new byte[]  
5         { 254, 241, 249, 224 }, 1);  
6     return recievedData;  
7 }
```

`public byte[] sendCommand(byte[] _command, int _respondSize)`

Tato funkce odešle senzoru příkaz specifikovaný parametrem `_command` a vyčká na doručení požadovaného množství dat. Ta jsou pak vrácena jako bajtové pole. Funkce je současně i přetížena a umožňuje vynechat parametr `_respondSize`.

Zdrojový kód 3.6: Odeslání příkazu senzoru

```
1 public byte[] sendCommand(byte[] _commandData, int  
2     _respondSize)  
3 {  
4     return m_connection.ExecuteCommand(0, _commandData,  
5         _respondSize);  
6 }
```

`public byte[] readData(int _respondSize)`

Tato funkce pouze přečte data aniž by nějaká data odeslala. To je výhodné zejména v případě, kdy chceme vyčíst všechna data, která má senzor v bufferu.

`public int BytesAvailable`

Proměnná udržující v sobě informaci o množství bajtů, které jsou v současné době v bufferu senzoru.

`public SensorDataFormat get1Data()`

Přečte právě jednu dávku měřených dat a předá je ve formátu definovaném třídou `SensorDataFormat` (viz. kapitola 3.3.2). Po přijetí měřených dat je zároveň aktivována událost `onMeasuredDataRecieve`, která umožňuje data číst nezávisle na měřené hodnotě. Tohoto se využívá například při kontinuálním čtení měření dat, kdy data nejsou vracena konkrétní metodě jako návratová hodnota, ale jsou čtena ve funkci upravující reakci na událost.

Zdrojový kód 3.7: Čtení měřených dat

```
public SensorDataFormat get1Data()  
2 {  
    byte[] recievedData = new byte[5];  
4    recievedData = m_connection.ExecuteCommand((byte)  
        CommandProtocol.Get_1Data, null, 5);  
  
6    SensorDataFormat sensorData = new SensorDataFormat(  
        recievedData);  
  
8    if (onMeasuredDataRecieve != null) onMeasuredDataRecieve(  
        sensorData);  
  
10    return sensorData;  
}
```

Pro přečtení většího množství měřených dat je připravena funkce `public byte getNData(int _data, CommandProtocol _com)`, která umožňuje vyčíst počet dat definovaný v kapitole 2.3.1.

`public byte[] getImage()`

Funkce umožňuje přečíst ze senzoru jeden údaj obrazových dat (viz. kapitola 2.3.2). Na začátku je zkontrolována šířka a výška parametru `Window`, která určuje velikost

obrazu a podle ní je určena výsledná velikost. Pro získání obrazu se čtvrtinovou velikostí je připravena funkce `public byte[] getImageReduced()`. Při přijetí jakéhokoliv typu obrazových dat je aktivována událost `OnImageRecieve`. Ta obdobně jako událost u měřených dat informuje o přijetí nového obrazu bez nutnosti číst data z návratové hodnoty.

`public ProfileData getProfile()`

Funkce pracuje obdobně jako funkce předchozí, ale tentokrát vyčítá údaje o profilových datech. Opět je na začátku nutné ověřit parametr `Window`. Tentokrát ale stačí informace o výšce. Pokud programátor z nějakého důvodu nechce data ve formě třídy `ProfileData` (viz. kapitola 3.3.4), může použít funkci `public byte[] getProfileBytes()`, která mu údaje vrátí jako bajtové pole.

`public EEPROMParameters getEEPROMParameters(byte _pageNum)`

Tato funkce přečte stránku paměti, která je určena parametrem `_pageNum`. Údaje jsou vráceny jako instance třídy `EEPROMParameters` (viz. kapitola 3.3.3). Ve funkci není kontrolována validita hodnoty `_pageNum`, protože může v budoucnu dojít k přidání nových stránek a obecnost této funkce by ztratila na významu. Podobnou funkcí, ale pro ethernetová data, je funkce `public EEPROMEthernet getEEPROMEthernet()`.

`public byte[] setEEPROMParameters(EEPROMParameters _eepromData, byte _pageNum)`

Následující funkce slouží naopak pro uložení EEPROM dat do zadané stránky EEPROM paměti. Konzistence dat je ověřena již při vytváření instance třídy `EEPROMParameters`, která je předaná jako parametr. Opět existuje stejná funkce i pro ethernetová data – `public byte[] setEEPROMEthernet(EEPROMEthernet _eepromData)`.

private int getValue(CommandProtocol _com)

Tato funkce je předpisem dalších funkcí, které čtou parametry senzoru (viz. zdrojový kód 3.8). Aby nebylo nutné pro každý parametr vytvářet samostatnou funkci, jsou funkce derivovány právě z této tím, že se předá název příkazu, čtoucí konkrétní parametr, jako parametr funkce. Pokud je z nějakého důvodu potřeba přecíst parametr jako pole bajtů, je možné využít funkci `private byte[] getValueBytes(CommandProtocol _com)`.

Zdrojový kód 3.8: Šablona funkce pro čtení parametrů

```
1 private int getValue(CommandProtocol _com)
2 {
3     byte[] recievedData = new byte[2];
4     recievedData = m_connection.ExecuteCommand((byte)_com,
5         null, 2);
6     int result = recievedData[0] * 256 + recievedData[1];
7     return result;
8 }
```

Ukázka čtení konkrétního parametru je pak ve zdrojovém kódu 3.9.

Zdrojový kód 3.9: Čtení parametru

```
1 public int getThreshold()
2 {
3     return getValue(CommandProtocol.Get_Threshold);
4 }
```

private byte[] setValue(CommandProtocol _com, int _value)

Princip definování předpisu funkce a její následné použití u parametrů je využít i při ukládání. Jako parametr funkce je nutné uvést příkaz, který konkrétní parametr definuje a vlastní hodnotu jako datový typ `int`. Funkce hodnotu převede na dva

bajty a ty otočí, protože vnitřní funkce jazyka C# ve výchozím nastavení pracuje s pořadím, kdy je spodní bajt první, v senzoru je to ale naopak.

Zdrojový kód 3.10: Šablona funkce pro uložení parametru

```
private byte[] setValue(CommandProtocol _com, int _value)
2 {
    byte[] valueBytes = BitConverter.GetBytes(_value);
4    Array.Reverse(valueBytes, 0, 2);

    byte[] recievedData = new byte[1];
    recievedData = m_connection.ExecuteCommand((byte)_com,
6        valueBytes, 1);
8    return recievedData;
}
```

3.4.1 Ostatní funkce třídy SensorClass

Třída SensorClass dále obsahuje jednodušší funkce k provedení základních operací. Mezi ně patří například funkce pro započítí a ukončení kontinuálního čtení měřených a profilových dat či funkce pro nastavení aktuálního režimu měření dat a používané stránky paměti EEPROM. Třída také obsahuje funkci `public string getFWVersion()`, která ze senzoru vyčte informaci o verzi firmwaru. Informace o verzi má ale předem danou strukturu, na kterou je nutné získanou hodnotu převést a vrátit jako řetězec.

3.5 Výsledná struktura rozhraní

Rozhraní, které bylo pojmenováno jako TLESensorClass tedy obsahuje následující třídy:

- Connections
 - Connection
 - Connection.LAN
 - Connection.USB

- Sensor
 - ActualSettings
 - CommandProtocol
 - EEPROMEthernet
 - EEPROMParameter
 - PointSize
 - ProfileData
 - **SensorClass**
 - SensorDataFormat
 - ValueHistory

4 Realizace ukázkové aplikace

Aby bylo možné ověřit schopnosti navrženého softwarového rozhraní v praxi, bylo nutné ho využít v rámci konkrétní aplikace. Aplikace byly nakonec vytvořeny dvě. První aplikace má sloužit hlavně jako demonstrace všech vlastností a schopností senzoru a s jejím využitím se počítá v rámci firmy Metralight. Aplikace byla nazvána TLE Studio.

Druhá aplikace, nazvaná Profile Scanner, vznikla za účelem demonstrace využití senzoru v rámci Technické univerzity v Liberci. Tato aplikace ukazuje použití senzoru při odměřování různých objektů jako jsou například desky plošných spojů, šrouby či kovové profily.

4.1 TLE Studio

Jak již bylo zmíněno výše, hlavním účelem aplikace je ukázat všechny schopnosti a vlastnosti senzoru TLE1. Bylo tak nutné vyřešit několik základních problémů. Prvním problémem je uživatelské rozhraní. Musí být natolik jednoduché, aby jej mohli ovládat i lidé, kteří nejsou programátoři a zároveň natolik komplexní, aby obsáhlo všechny funkce senzoru. Aplikace musí být současně natolik robustní, aby neskončila chybou například v případě, že uživatel zadá špatné údaje do polí pro vstup textu – bylo tedy zásadní, aby byly ošetřeny všechny chybové stavy.

Druhým problémem bylo navržení vhodné reprezentace měřených dat. U měřené hodnoty je situace jasná – jedná se pouze o jedno číslo. U profilových dat a dat obrazových je ale situace jiná, protože data lze vizualizovat různými způsoby.

4.1.1 Uživatelské rozhraní

Uživatelské rozhraní je základní a nejdůležitější částí aplikace. Právě skrze něj uživatel komunikuje se senzorem a díky němu může nastavovat údaje nebo naopak jiná data číst. První návrhy byly realizovány pomocí C# frameworku Windows Forms. Protože ale bylo nutné s uživatelským rozhraním pracovat více detailně, bylo rozhodnuto o přechodu na jiný framework – konkrétně framework WPF. Ten umožňuje například jednodušší změnu barev, grafické efekty nebo stínování. Jeho výhodou je současně i to, že uživatelské rozhraní definuje v XML souboru. Je tak oddělena funkční část aplikace od grafického návrhu. Microsoft pro jednodušší práci s WPF dodává i samostatnou aplikaci Blend.

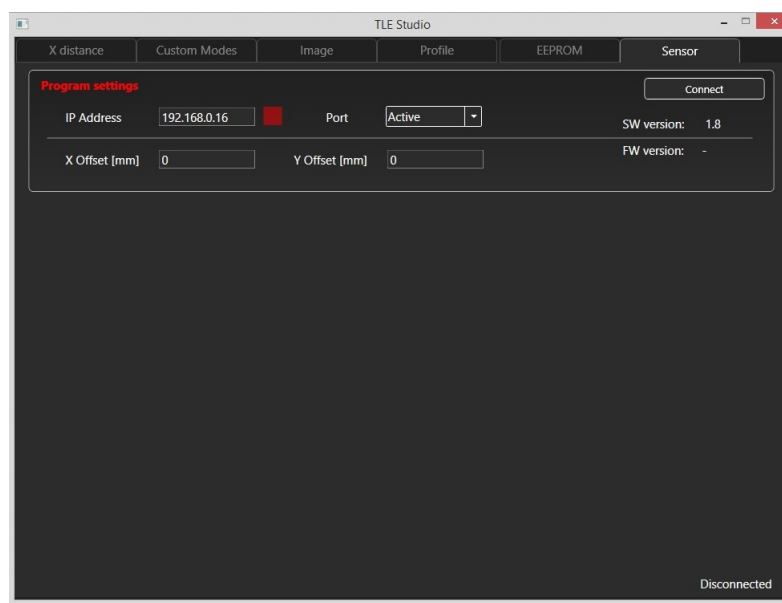
První vlastností uživatelského rozhraní aplikace TLE Stuido, která byla řešena, je velikost okna. Protože je možné, že se aplikace bude spouštět i na průmyslových počítačích, bylo by nevhodné použít velké okno. Bylo proto rozhodnuto, že maximální velikost okna bude 1024 x 768 pixelů. Aplikace je však ještě o něco menší, protože se musí počítat i s velikostí nabídky Start operačního systému Windows.

Druhou důležitou vlastností je barva okna a celkové barevné schéma aplikace. První návrhy měly bílé pozadí a černý text. To se však později ukázalo jako nevhodné, protože delší sledování obrazovky monitoru s bílým pozadím velmi namáhá oči. Výsledné schéma je tedy tmavé se světlým textem. Výhodou je současně i to, že v tmavé barvě více vyniknou další barvy jako červená nebo zelená. Současně se změnou barevného schématu bylo ale nutné znovu navrhnout grafickou úpravu všech ovládacích prvků, protože výchozí nastavení WPF je černý text na bílém pozadí. Výsledné grafické schéma je zobrazeno na obrázku 4.1.

Po grafické stránce aplikace bylo nutné navrhnout i její logickou strukturu, tedy oddělit například měřená data od dat obrazových. Pro tyto účely byla aplikace rozdělena do několika záložek:

- **X Distance** – záložka pro základní čtení měřených hodnot ve výchozím režimu
- **Custom Modes** – záložka pro čtení měřených hodnot v dalších režimech

- **Image** – záložka pro zobrazování obrazových dat a nastavení souvisejících parametrů
- **Profile** – záložka pro čtení profilových dat
- **EEPROM** – záložka pro práci s pamětí
- **Sensor** – záložka pro připojení k senzoru a nastavení dočasných parametrů



Obrázek 4.1: Okno aplikace TLE Studio

Protože se v aplikaci předpokládá velké množství údajů zadávaných uživatelem, bylo nutné vytvořit jednotný způsob validace těchto dat. Pro tyto účely byla standardní komponenta `TextBox`, která slouží k zadávání uživatelského vstupu, rozšířena o nové funkce. Nově tedy pole umožňuje zadat typ hodnot, které do něj budou vkládány a na základě tohoto údaje ověřit, zda jsou data správná. Pole podporuje následující vstupy:

- **NumberDecimal** – celočíselný vstup
- **NumberFloat** – vstup čísel s plovoucí řádovou čárkou
- **Boolean** – hodnoty `true` a `false` (nebo i 1 a 0)
- **IP String** – řetězec specifikující adresu IP
- **Text** – klasický řetězec

Pro validaci pak stačí použít funkci `checkValue()`, která hodnoty ověří. V případě, že jsou údaje zadány chybně, je uživatel na tuto skutečnost upozorněn zobrazením bílého křížku vedle pole a obarvením hodnoty na červenou. V případě korektního zadání hodnot se grafická úprava pole nemění. Třída upraveného pole `TextBox` byla nazvána **`TextBox_Improved`**.

Důležitou funkcí aplikace je časovač. Ten se stará o to, aby byly v pravidelných intervalech čteny údaje o měřené hodnotě, obrazu a profilu a to v závislosti na právě aktivované záložce. Při přepnutí do záložek `Sensor` nebo `EEPROM` je časovač zastaven, protože v těchto záložkách není nutné periodicky číst hodnoty.

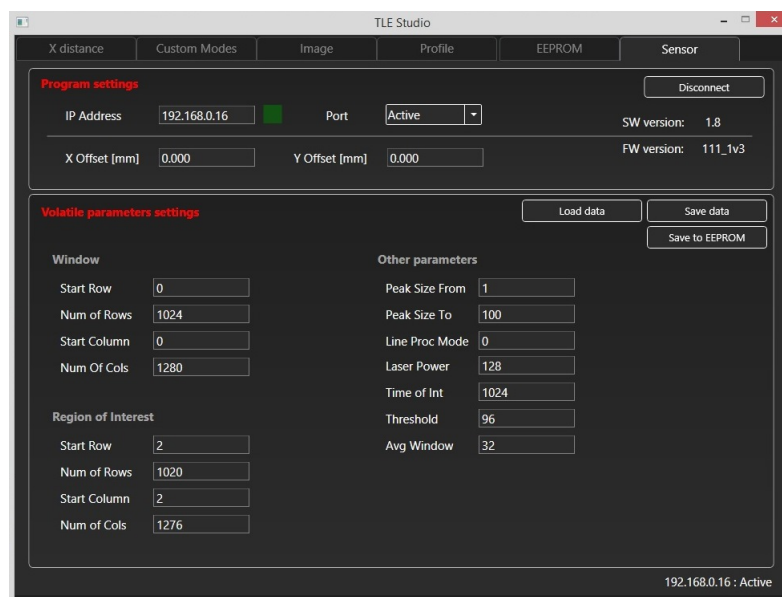
4.1.2 Záložka `Sensor`

Záložka `Sensor` je výchozí záložkou, která se uživateli aktivuje po spuštění aplikace. Bez připojení k senzoru není možné tuto záložku opustit a současně nejsou bez připojení zobrazeny všechny ovládací prvky. Stránka obsahuje základní prvky pro připojení k senzoru - IP adresu a port. Senzor umožňuje připojení ke dvěma portům - 1024 a 1028. Protože by ale bylo pro uživatele zbytečné si porty pamatovat, byly nahrazeny slovním ekvivalentem `Active` a `Pasive`. Po připojení k portu `Active` je uživateli dovoleno pracovat se všemi funkcemi senzoru. Připojí-li se ale uživatel k portu `Pasive`, nemůže v senzoru nic nastavovat ale pouze číst. Dalšími prvky na stránce jsou informace o verzi firmwaru, verzi softwaru a pole pro nastavení offsetu hodnot `X` a `Y`. Do nich je možné zadat jak kladná, tak záporná čísla a bude použit v rámci celé aplikace. Všechny doposud zmíněné prvky jsou sjednoceny do panelu s názvem `Program settings`.

Dalším panelem na stránce, který se ale zobrazí až po připojení k senzoru, je panel `Volatile parameter settings`, který vypisuje všechny údaje uložené v dočasné paměti senzoru. Zde je možné hodnoty jak číst, tak měnit a zapisovat do paměti. Je zde i tlačítko pro zkopírování všech parametrů do `EEPROM` paměti. Po jeho stisknutí se zobrazí nové okno, které uživatele vyzve k zadání stránky paměti, do které mají být hodnoty uloženy. Všechna vstupní pole jsou typu `TextBox_Improved`.

Před uložením jsou tedy hodnoty ověřeny a v případě chybného zadání není uložení umožněno. Po připojení k senzoru je v dolní části okna zobrazena informace o IP adrese senzoru a konkrétním portu. Tato informace je viditelná při aktivaci jakékoliv záložky.

Podoba záložky v případě nepřipojení k senzoru je na obrázku 4.1, připojenou záložku pak zobrazuje obrázek 4.2.



Obrázek 4.2: Záložka Sensor aplikace TLE Studio

4.1.3 Záložka X Distance

Hlavním účelem této záložky je poskytnout uživateli informaci o měřené hodnotě v režimu měření MODE_0 (viz. 2.1). V horní části okna je tak zobrazena velkým písmem samotná hodnota s jednotkou spolu s maximem a minimem. Tyto dva údaje lze resetovat kliknutím na jakékoliv číslo či stisknutím klávesy R. Ve spodní části okna pak nabíhá graf reprezentující historii s časem na ose X a hodnotami na ose Y. Graf je vyplněn červenou barvou s mírnou průhledností, aby byly stále čitelné popisky os. Obrázek 4.3 zobrazuje podobu záložky.

Při zobrazování měřené hodnoty nastal problém s použitím tečky nebo čárky jako desetinného oddělovače. Jazyk C# se ve výchozím nastavení přizpůsobí jazyku systému. Tato vlastnost má ale za následek nepředvídatelné chování na anglické verzi systému Windows, protože se nastavení projevuje například i při převádění hodnot z řetězců na čísla. Bylo proto rozhodnuto, že se v programu napevno nastaví tečka jako desetinný oddělovač. Program se tak chová stejně při použití jakékoliv jazykové mutace systému Windows.

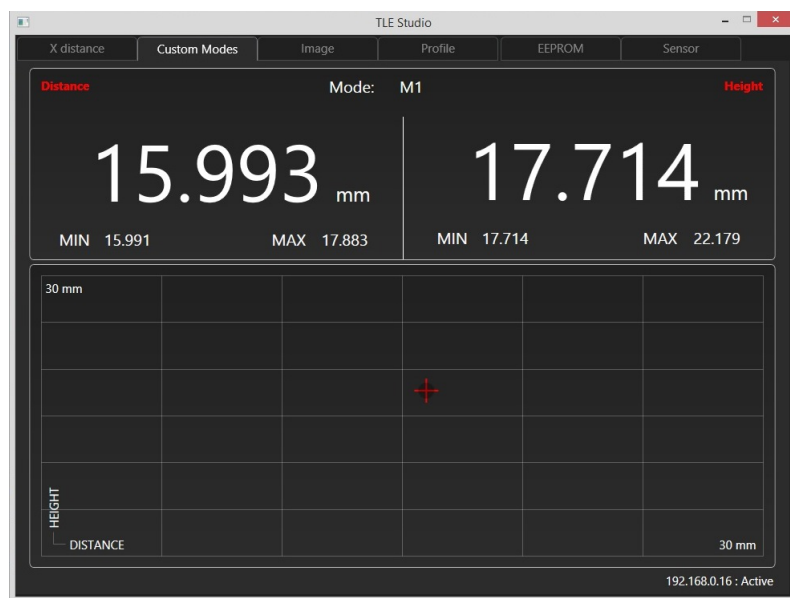


Obrázek 4.3: Záložka X Distance aplikace TLE Studio

4.1.4 Záložka Custom Modes

Tato záložka slouží k zobrazování měřených dat v jiných režimech než je režim MODE_0. Kromě vzdálenosti tedy zobrazuje i údaj o výšce. Oba údaje jsou zároveň doplněny svojí maximální a minimální hodnotou, kterou lze opět resetovat stiskem klávesy R nebo klepnutím na jakékoliv pole. Ve výchozím nastavení senzoru jsou dostupné režimy měření MODE_1 až MODE_3. K přepnutí režimu stačí kliknout na název režimu v horní části obrazovky.

Pro zlepšení přehlednosti a větší názornost toho, co je vlastně měřeno, byl ve spodní části záložky vytvořen graf se vzdáleností na ose X a výškou na ose Y. V tomto grafu je zakreslen bod korespondující s aktuálně měřenou hodnotou. Aby byly trošku naznačeny i minulé hodnoty, má tento bod „ocas“, který je právě na základě předchozích hodnot vykreslován. Podoba záložky je na obrázku 4.4,



Obrázek 4.4: Záložka Custom aplikace TLE Studio

4.1.5 Záložka Image

Záložka Image slouží k zobrazení obrazových dat senzoru. Záložka je rozdělena na několik částí. V levé části je zobrazen samotný obraz, pod ním pak graf řezu označeného řádku a pravá část je vyhrazena pro nastavení parametrů.

Funkce vykreslování obrazu se skládá z několika částí. První částí je převod bajtového pole na bitmapu, která je pak vykreslována. Jazyk C# ale bohužel nemá vestavěnou funkci, která tento problém přímo řeší. Bylo proto nutné napsat funkci, která tento převod zajistí. Protože jsou ale obrazy vykreslovány rychle za sebou, bylo nutné při vytváření myslet i na rychlost a optimalizaci. Pro tyto účely bylo přikročeno k využití ukazatelů (Herout (2009)). Ty pracují v tzv. „unsafe“ bloku

kódu, který není při kompilaci striktně kontrolován. To má však za nevýhodu to, že může snadno dojít k přepsání paměti, která přepsána být nemá. Microsoft sám upozorňuje na únik paměti, který u této funkce vzniká. Bylo proto nutné nenechávat úklid paměti plně na garbage collectoru, ale mazat paměť manuálně. Výsledkem je funkce, která vykreslí snímek při rozlišení 640 x 512 pixelů za méně než 10 ms. Druhým problémem při vykreslování obrazu je to, že obraz může mít různou velikost v závislosti na nastavení parametru Window. Je tedy nutné obraz po převodu na bitmapu přesunout na správnou pozici.

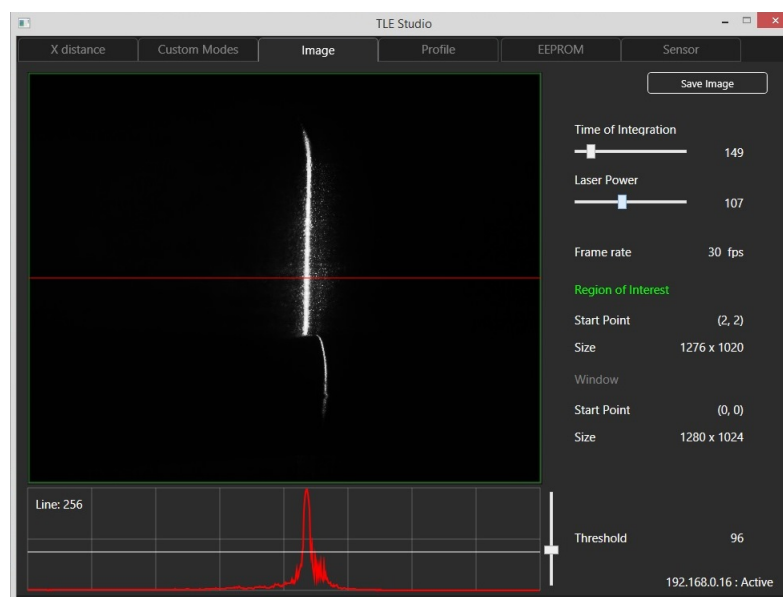
Další funkcí, kterou má záložka Image na starosti, je nastavování velikostí oblastí Window a Region of Interest. Jak bylo popsáno v kapitole 3.3.5, jedná se o obdélníky specifikující aktivní oblast, kde senzor měří resp. zpracovává data. Pro jejich nastavení existuje několik možností. Nejjednodušší možností by bylo použít klasická pole a šířku, výšku a počáteční bod do nich zapsat. Toto by ale uživateli znemožnilo vizuálně zkontrolovat správnost zadaných údajů. Další možností proto je použití posuvníků vedle obrazu. Tato komponenta umožňuje pomocí jezdců nastavit hodnotu a jeden posuvník může mít jezdců více. Nebyl by tak problém nastavit oblast s použitím dvou posuvníků. Toto řešení ale nevypadá vizuálně hezky a ergonomie ovládání není ideální. Bylo proto přikročeno k vytvoření komplexní grafické komponenty, která vykreslí oblasti Window a ROI přímo do obrazu a jejich změnu umožňuje tažením za okraje obdélníku. Uživatel tak má okamžitou vizuální zpětnou vazbu o tom, jak parametry nastavil.

S obrazem souvisí i graf řezu aktivního řádku. Tento graf je umístěn přímo pod obrazem a po kliknutí na libovolný řádek ho vykreslí jako graf s pixely na ose X a jejich hodnotou na ose Y. Tento graf slouží primárně pro jednodušší nastavení parametru Threshold – tedy hodnoty, která specifikuje minimální hodnotu pro zpracování dat. Threshold je v grafu vykreslen jako bílá čára a vizuálně uživatele informuje o svojí hodnotě. Tu je možné měnit pomocí posuvníku napravo od grafu.

Zobrazení obrazových dat i nastavení oblastí bylo spojeno do jedné vizuální komponenty, která má několik vrstev. Ve spodní vrstvě se vykresluje samotný obraz. Ten

je pozicován v rámci černého okna. Nejsou tak vidět okraje a obraz působí, jako by byl stejně tak velký jako komponenta samotná. Vyšší vrstva obsahuje obdélníky pro nastavení oblastí Window a ROI. Bylo však nutné vyřešit situaci, kdy jsou obdélníky blízko sebe a bylo by nesnadné trefit se na čáru konkrétní oblasti. Obdélník je proto nutné předem aktivovat kliknutím na jeho název v pravé části okna. Aktivní obdélník má zelenou barvu, neaktivní šedou a pouze s aktivním obdélníkem je možné manipulovat. V nejvyšší vrstvě komponenty je pak čára řezu. S tou je možné pohybovat za jakékoliv situace.

Kromě výše zmíněných údajů je v rámci záložky Image možné nastavit i parametr dobu integrace (Time of Integration) a výkon laseru (Laser Power). K těmto účelům slouží posuvníky v pravé horní části okna. Jejich změna se vždy projeví až po uvolnění tlačítka myši. Obraz je možné také uložit do souboru stisknutím tlačítka Save Image. Na disku je pak vytvořen BMP soubor, jehož název obsahuje časový otisk okamžiku, kdy byl vytvořen. Výsledná podoba záložky Image je zobrazena na obrázku 4.5.

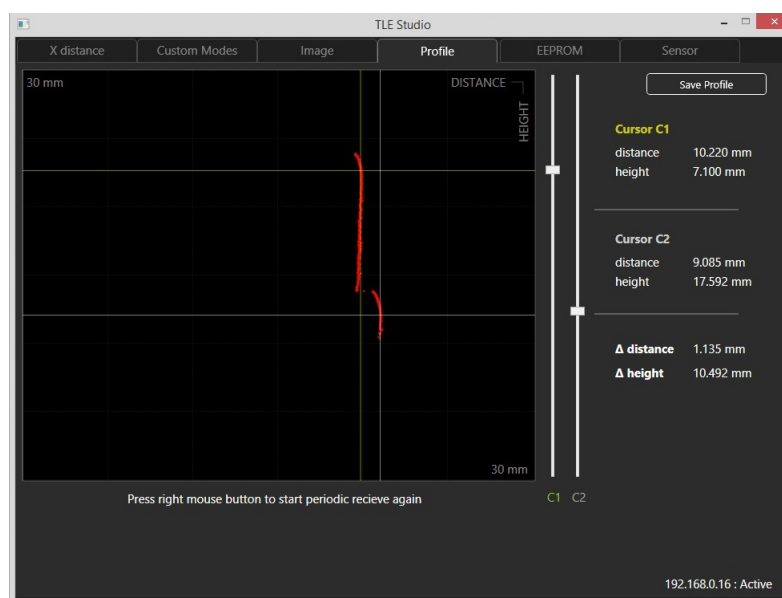


Obrázek 4.5: Záložka Image aplikace TLE Studio

4.1.6 Záložka Profile

Tato záložka slouží k zobrazení profilových dat, tedy předzpracovaných obrazových dat. Hlavním prvkem záložky je graf zobrazující profilová data. Graf reprezentuje soubor dat, který je čten ze senzoru ve formě bajtového pole. Toto pole obsahuje vždy údaj o vzdálenosti a hned za ním údaj o výšce. Je tak vytvářen dvourozměrný graf se vzdáleností na ose X a výškou na ose Y. I u tohoto grafu bylo potřeba vyřešit problém s pozicováním výsledku. Tentokrát ale výsledná pozice není závislá na celé oblasti Window, ale pouze na jeho výšce.

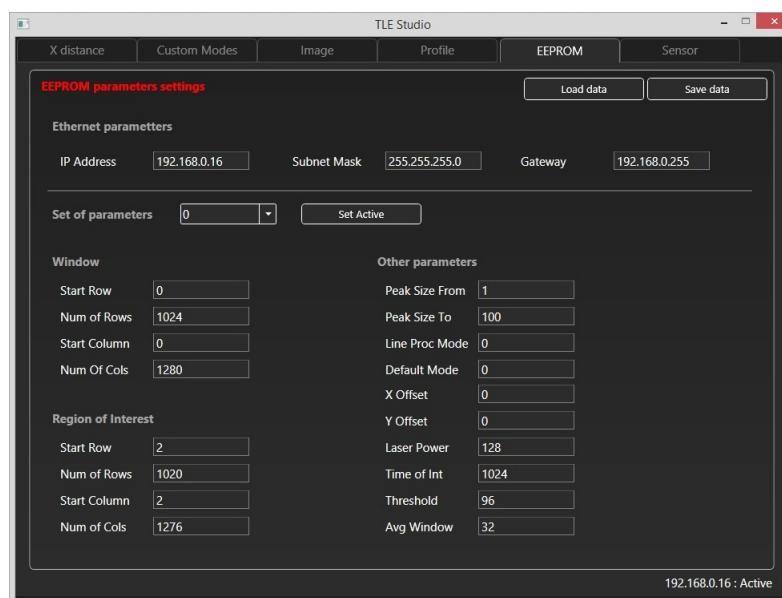
Protože se jedná o dvourozměrné pole reprezentující sadu měřených dat, byla uživateli poskytnuta možnost změřit hodnoty vzdálenosti a výšky v jednotlivých bodech použitím posuvníků na pravé straně. Při přesunutí posuvníku do oblasti měřených dat jsou vykresleny dvě čáry, které se protnou právě v bodě, který chce uživatel změřit. Použitím druhého posuvníku lze změřit i rozdíl hodnot mezi dvěma body. Pro lepší přehlednost je jeden posuvník bílý a druhý žlutý. Výslednou podobu záložky Profile zachycuje obrázek 4.6.



Obrázek 4.6: Záložka Profile aplikace TLE Studio

4.1.7 Záložka EEPROM

Hlavním účelem záložky EEPROM je umožnit uživateli načítat a ukládat data z a do vnitřní paměti senzoru. Po aktivaci záložky jsou do všech polí načteny hodnoty aktuálně zvolené stránky paměti EEPROM. Údaje lze měnit, uložit a následně i manuálně znovu načíst. Všechna pole jsou typu TextBox_Improved. Data jsou tedy před uložením validována. V rámci této záložky lze současně i nastavit aktuálně používanou stránku paměti pomocí tlačítka Set Active. Podoba záložky EEPROM je na obrázku 4.7.



Obrázek 4.7: Záložka EEPROM aplikace TLE Studio

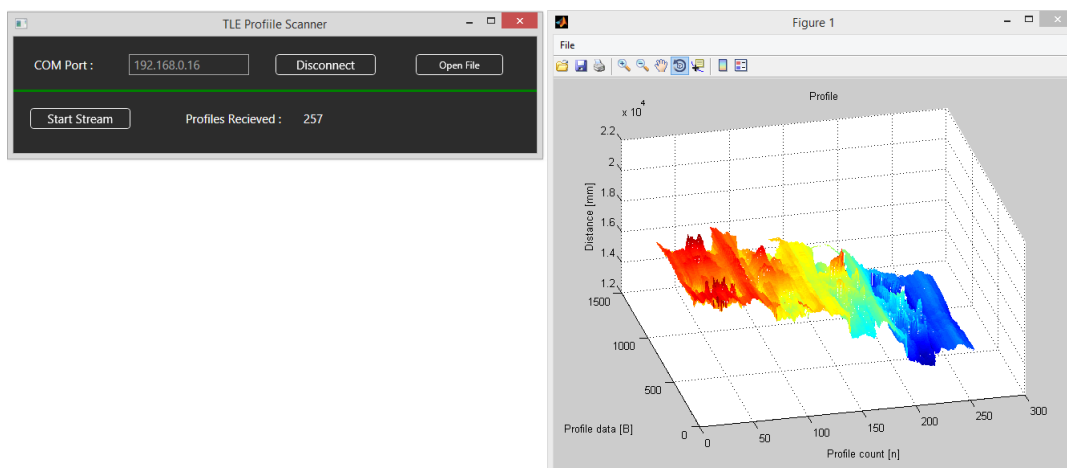
4.2 Profile Scanner

Aplikace Profile Scanner byla vytvořena za účelem prezentace možností senzoru v rámci Technické univerzity v Liberci. Tato aplikace by mohla být začleněna do TLE Studia, ale firma Metralight tuto funkcionalitu nepotřebuje. Bylo proto rozhodnuto o oddělení této části do samostatné aplikace.

Aplikace je nazvána Profile Scanner, protože využívá profilová data senzoru, ze kterých vytváří prostorový obraz objektu. S profily na ose X, bajty profilu na ose Y a vzdáleností na ose Z.

Senzor právě pro využití v podobných případech umožňuje kontinuální streamování dat (viz. kapitola 2.3.3). Zasláním příkazu pro start streamu začne senzor posílat profily tak rychle, jak sám zvládne. Jediné omezení rychlosti by tak mohlo nastat na straně počítače, který profilová data zpracovává. Aby k tomu však nedocházelo, jsou profilová data vyčítána ze senzoru a ukládána do zásobníku. Data jsou zpracována až v okamžiku, kdy uživatel stiskne tlačítko pro konec streamování dat.

Výslednou podobu programu zobrazuje obrázek 4.8. Aplikace samotná je tvořena pouze jedním oknem, ve kterém jsou prvky pro připojení k senzoru a spuštění kontinuálního streamu dat. Postup streamu je indikován popiskem, který zobrazuje počet aktuálně načtených streamů. Chce-li uživatel vizualizovat předem uložený soubor profilových dat, může využít tlačítko Open File.

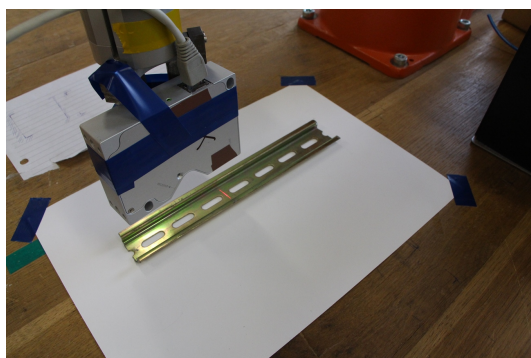


Obrázek 4.8: Aplikace Profile Scanner

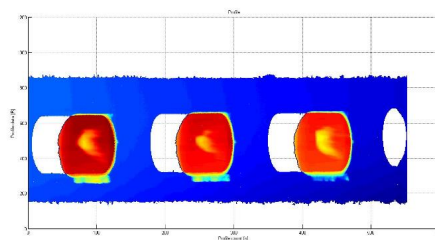
První návrhy vizualizace profilových dat počítaly s vytvořením funkce, která tato data vykreslí. Protože by ale bylo velmi složité vytvořit funkci, která by data zobrazila prostorově, byla vytvořena funkce, která profilová data zobrazovala pouze

dvourozměrně. Rozdílné vzdálenosti pak byly odlišeny různou barvou. Toto řešení se však ukázalo jako málo názorné a pro další využití nešikovné. Po prostudování několika grafických frameworků jako je například AForge.NET bylo rozhodnuto, že se pro vykreslení grafu využije prostředků programovacího prostředí Matlab, který má vykreslovací a modelovací funkce na velmi pokročilé úrovni. S objektem je možné otáčet, zvětšovat a zmenšovat velikost a výsledek je možné uložit do několika grafických formátů. Matlab podporuje načítání DLL knihoven vytvořených programovacím jazykem C# .NET. Není tak problém v Matlabu knihovnu použít. Program Profile Scanner tedy pracuje tak, že data načítá a ukládá program napsaný v jazyce C# a až při vizualizaci je zavolána funkce prostředí Matlab.

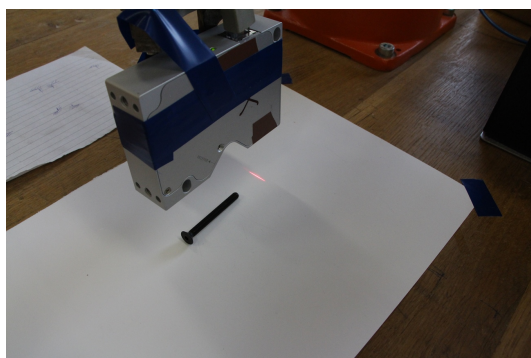
V rámci hledání možnosti využití senzoru pro účely Technické univerzity v Liberci bylo nasnímáno několik různých objektů. Senzor je například možné použít pro posuzování kvality plošných spojů a osazení součástkami. Zde se ale projevuje limitace principu senzoru. Senzor měří vzdálenost vysíláním laserového paprsku. Pokud je tedy deska plošných spojů lakována nebo se měď leskne, senzor není schopen přesně měřit, protože měření ovlivňují odrazy. Další možné využití je například při posuzování kvality závitů šroubů. Tam senzor vrací dobré výsledky, protože se nejedná o souvislou lesklou kovovou plochu. Na výsledném grafu jsou tedy vidět závity a v případě vady by se v grafu jasně projevily. Kromě těchto dvou případů lze senzor využít pro měření otvorů v různých materiálech či reliéfu. Výsledná měření jsou na obrázku 4.9



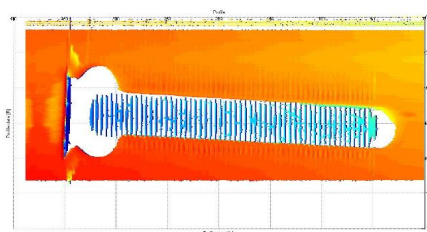
(a) Měření kovového profilu



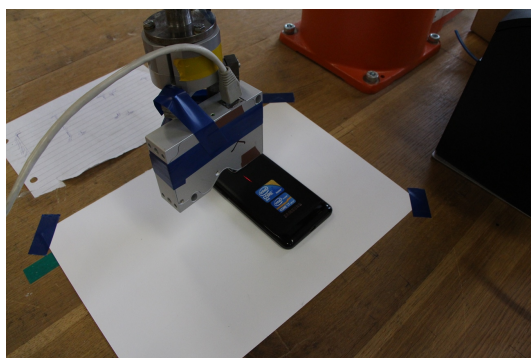
(b) Výsledný profil



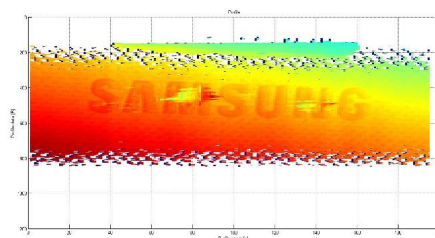
(c) Měření šroubu



(d) Výsledný profil



(e) Měření zadní strany mobilního telefonu



(f) Výsledný profil

Obrázek 4.9: Nasnímané objekty programem Profile Scanner

5 Závěr

V diplomové práci se podařilo splnit všechny body zadání. Jako první byl prostudován princip práce senzoru TLE1 a jeho komunikační protokol. Následně bylo realizováno softwarové rozhraní. Toto rozhraní bylo navrženo natolik variabilně, že je v budoucnu snadno rozšiřitelné o nové komunikační rozhraní, funkce a parametry senzoru. Rozhraní zároveň poskytuje veškerou funkcionalitu senzoru a pro programátory tak představuje snadný prostředek pro tvorbu vlastních aplikací. Pro prověření schopností rozhraní byly vytvořeny dvě aplikace. První aplikace poskytuje obecný přehled o funkcích senzoru díky vizualizaci jeho dat. Uživatel tak má snadný přístup k měřeným hodnotám a obrazovým a profilovým datům. Zároveň může snadno pracovat s vnitřní pamětí a nastavovat parametry senzoru.

Rozhraní je možné v budoucnu dále rozšířit například přidáním připravených grafických komponent, které bude moci uživatel jednoduše zařadit do svého programu. Je také možné do rozhraní začlenit další senzory firmy Metralight, protože většina z nich sdílí podobné komunikační rozhraní a liší se pouze v parametrech. V aplikaci Profile Scanner by do budoucna bylo možné začlenit funkce pro měření údajů přímo v obraze větším využitím programovacího prostředí Matlab.

Literatura

AForge.NET. *AForge.NET Framework* [online]. [cit. 8.5.2014]. Dostupné z:
<http://www.aforogenet.com/framework/>.

GAMMA, Erich et al., 1994. *Design Patterns: Elements of Reusable Object-Oriented Software*. Boston : Addison-Wesley. ISBN 0-201-63361-2.

HEROUT, Pavel, 2009. *Učebnice jazyka C*. České Budějovice : Kopp. ISBN 978-80-7232-383-8.

KOTTNAUER, Jakub. *Vlákna v C#* [online]. [cit. 8.5.2014]. Dostupné z:
<http://programujte.com/clanky/35-vlakna-v-c/>.

METRALIGHT, 2012. *TLE1 User's Guide*. Liberec. Dostupné z:
http://www.metrallight.com/products/doc/tle1/tle1_ug_d.pdf.

MICROSOFT. *Microsoft Developer Network* [online]. [cit. 8.5.2014]. Dostupné z:
<http://msdn.microsoft.com/en-US/>.

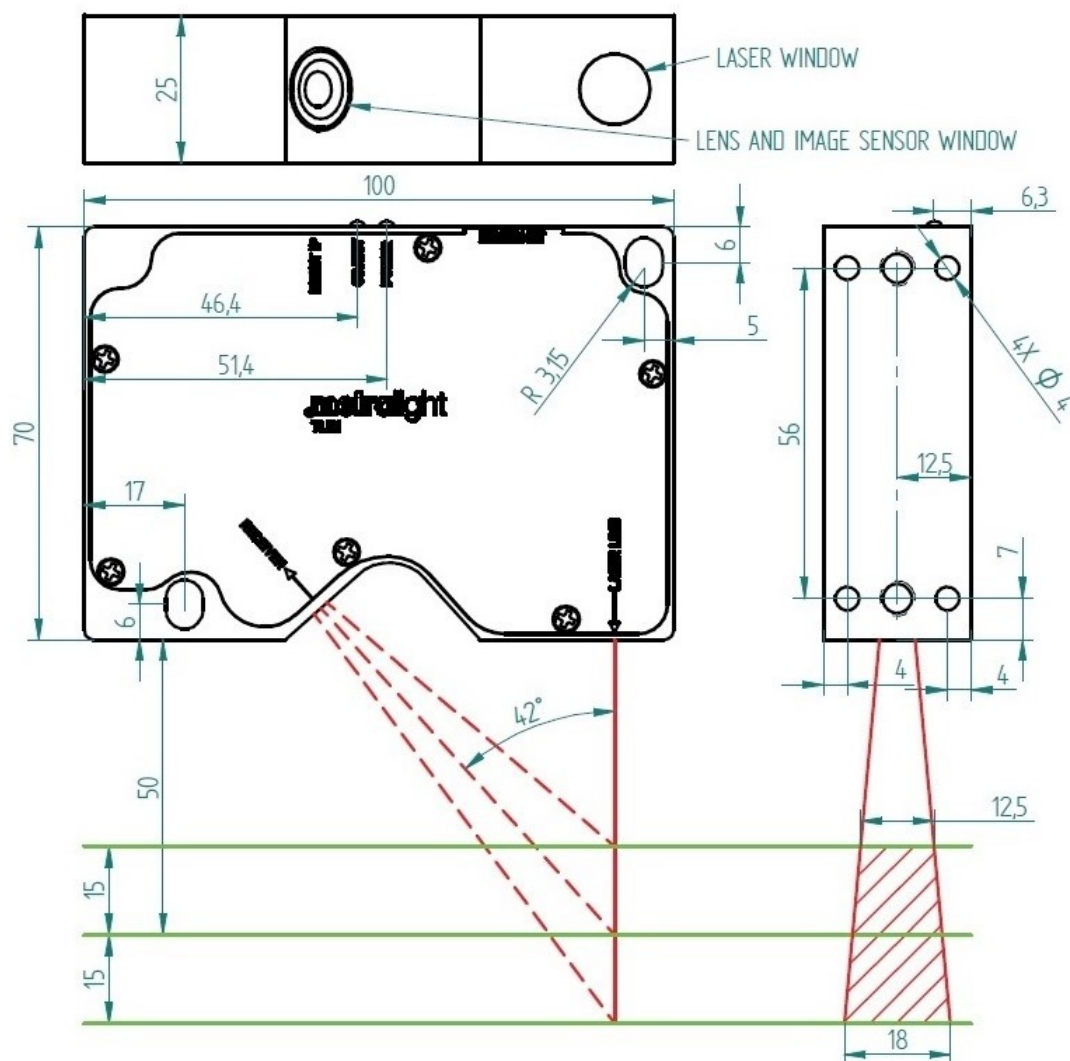
NAGEL, Christian, EVJEN, Bill, 2009. *C# 2008: Programujeme profesionálně*. Brno : Computer Press, 1. vyd. ISBN 978-80-251-2401-7.

PRATA, Stephen, 2004. *Mistrovství v C++*. Brno : Computer Press. ISBN 978-80-251-0098-7.

SHARP, John, 2010. *Microsoft Visual C# 2010: Krok za krokem*. Brno : Computer Press, 1. vyd. ISBN 978-80-251-31473.

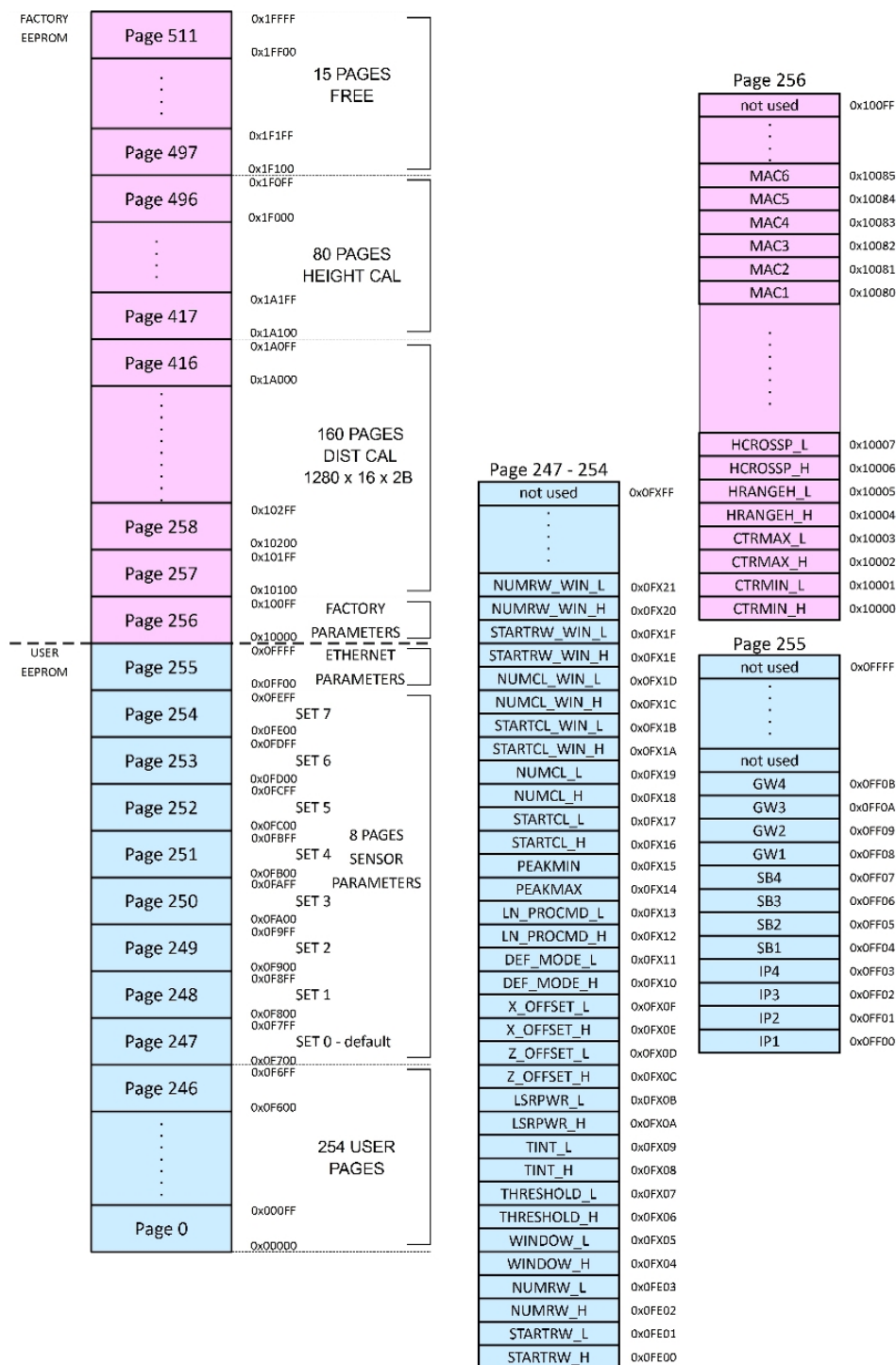
STACK EXCHANGE. *Stack Overflow* [online]. [cit. 8.5.2014]. Dostupné z:
<http://stackoverflow.com/>.

A Rozměry senzoru TLE1



metralight

B EEPROM mapa



C Volatilní parametry senzoru

Parametr	Popis	Čtení	Zápis	Rozsah
ROI_START_COLUMN	První sloupec ROI	0x50	0x58	2 – 1248
ROI_NUM_OF_COLUMNS	Počet sloupců ROI	0x51	0x59	32 – 1278
WIN_START_LINE	První řádek Window	0x52	0x5A	1 – 1020
WIN_NUM_OF_LINES	Počet řádků Window	0x53	0x5B	4 – 1024
WIN_START_COLUMN	První sloupec Window	0x54	0x5C	1 – 680
WIN_NUM_OF_COLUMNS	Počet sloupců Window	0x55	0x5D	600 – 1280
ROI_START_LINE	První řádek ROI	0x80	0x88	2 – 1022
ROI_NUM_OF_LINES	Počet řádků ROI	0x81	0x89	2 – 2022
LINE_PROCMODE	Režim zpracování	0x86	0x8E	0 – 3
PEAK_SIZE	Hodnota zpracování	0x87	0x8F	1 - 65535

D Obsah vloženého CD

Vložené CD obsahuje kompletní zdrojové kódy rozhraní, aplikací TLE Studio a Profile Scanner a samotného textu práce a dokumentace, které byly zpracovány v L^AT_EX.

- `text_documentation` – zdrojové kódy dokumentace
- `text_thesis` – zdrojové kódy textu diplomové práce
- `TLEProfileScanner` – zdrojové kódy aplikace Profile Scanner
- `TLESensorClass` – zdrojové kódy vytvořeného softwarového rozhraní
- `TLEStudio` – zdrojové kódy aplikace TLE Studio
- `Nemecek_DP_2014.pdf` – výsledný PDF soubor diplomové práce

E Dokumentace k rozhraní TLESensorClass

TLE Sensor DLL Class User's Guide

May 12, 2014

1 Setting up Visual Studio for using TLE Sensor DLL Class

1.1 C# .NET

TLE Sensor library is written in C# programming language and uses .NET Framework 4.5. This guide was made for Visual Studio 2012 and 2013.

1. Open Visual Studio and create new project in **File** → **New** → **Project**
2. From menu on left side select **Templates** → **Visual C#** → **Console Application**, enter the name, location and press **OK**

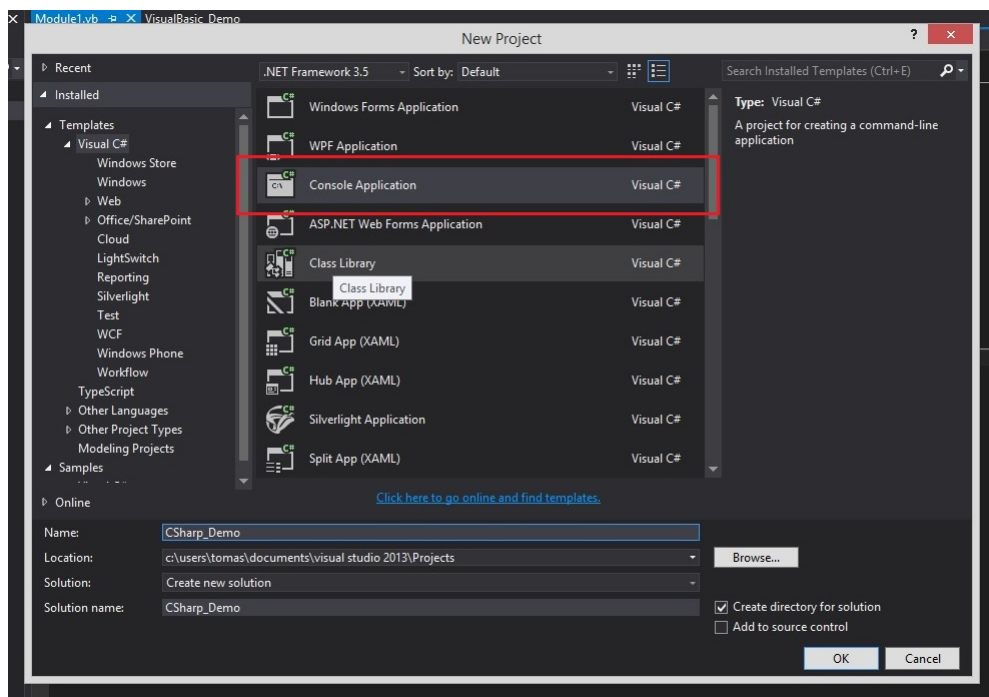


Figure 1: Creating a new project

3. When the new project is created, we need to add a reference to TLE Sensor DLL class. In Solution Explorer menu right click on **References** and press **Add Reference...**

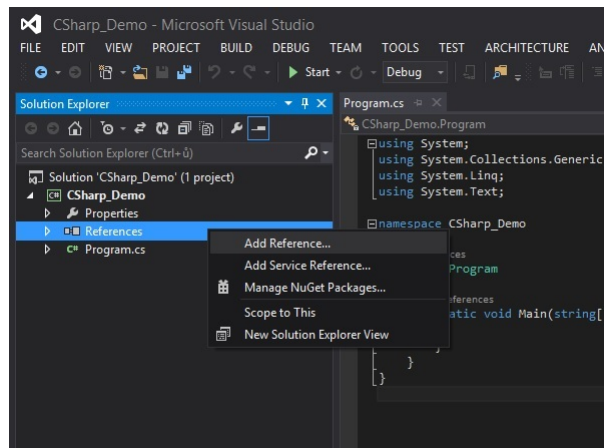


Figure 2: Reference menu

4. In Reference Manager menu select **Browse**, find TLESensorClass.dll and press **OK**. Check if you can see TLESensorClass in Reference menu in Solution Explorer.

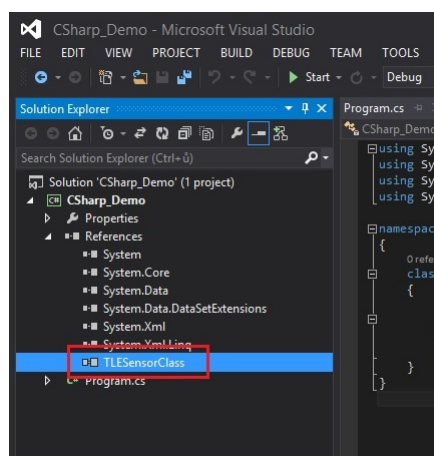


Figure 3: Reference checking

5. Now we need to set the target framework of the project to .NET Framework 4.5. In Solution Explorer right click on the project name and select **Properties**. In Properties menu set the **Target framework** to **.NET Framework 4.5**.

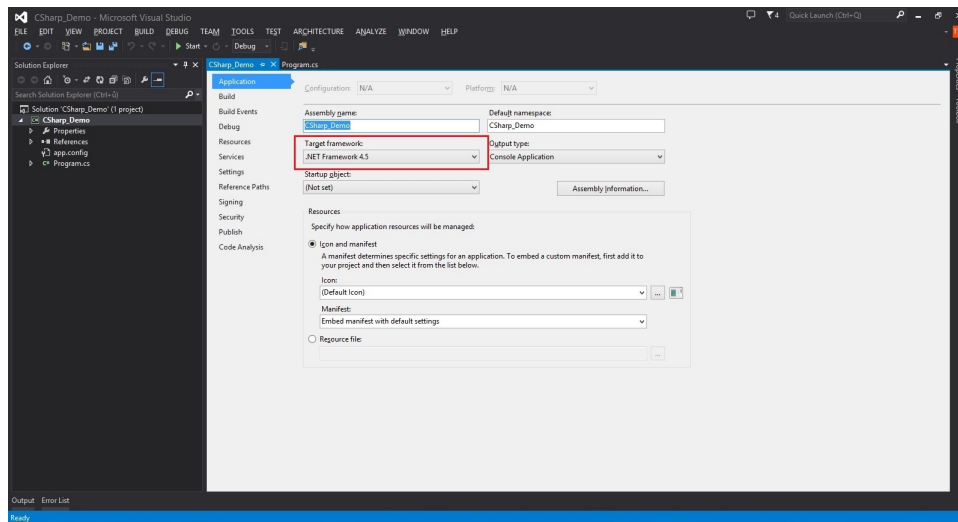


Figure 4: Properties Window

Now, when the TLE Sensor DLL class was linked, we can start writing an application which will use the TLE Sensor.

Application example

In this sample application we will just connect to sensor, read measurements and print them to console.

```
using System;

// Necessary for using TLE Sensor DLL classes
using TLESensorClass.Connections;
using TLESensorClass.Sensor;

namespace CSharp_Demo
{
    class Program
    {
        static void Main(string[] args)
        {
            // Definition of connection information – IP ↔
            // address and port
            Connection con = new Connection_LAN("↔
            192.168.0.20", 1024);
            // Creating sensor object with defined ↔
            // connection data
            SensorClass sensor = new SensorClass(con);
            // connecting to the sensor
            sensor.connect();

            // reading measurements and printing them to ↔
            // console
            Console.WriteLine(sensor.get1Data());
        }
    }
}
```


1.2 Visual Basic .NET

Since Visual Basic .NET uses the same internal structure of it's code, it's very easy to import C# DLL into the project.

1. Open Visual Studio and create a new project in **File** → **New** → **Project**
2. From the menu on left side select **Templates** → **Other Languages** → **Visual Basic** → **Console Application**, enter the name, location and press **OK**.

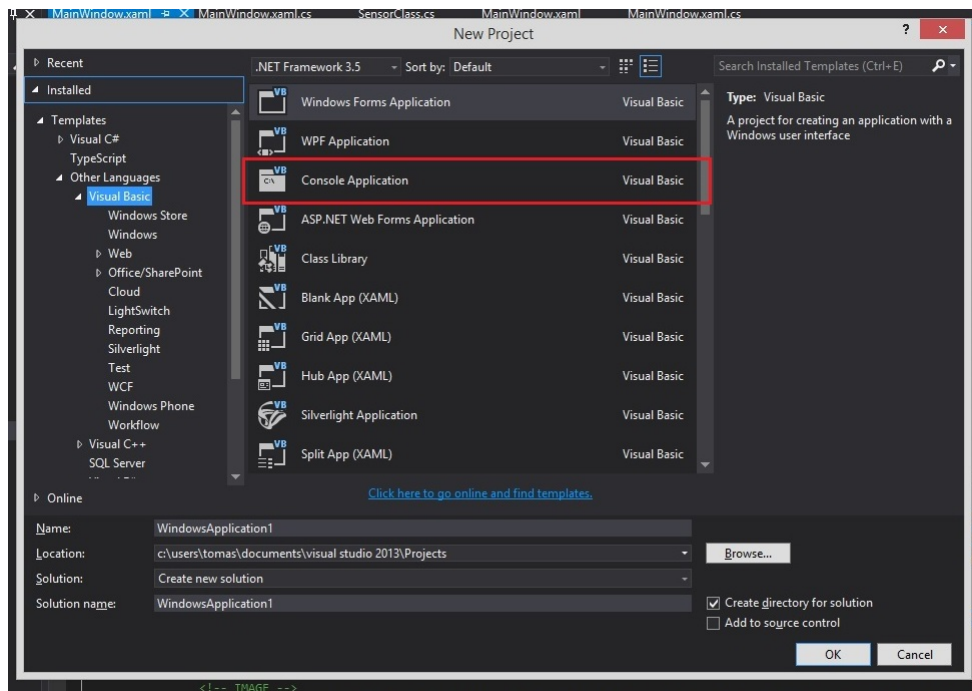


Figure 5: Creating of new project

3. When the new project is created, we need to add a reference to the TLE Sensor DLL class. In Solution Explorer right click on project name and press **Properties**. In the opened tab click on **References** and press **Add** in the middle of the tab.

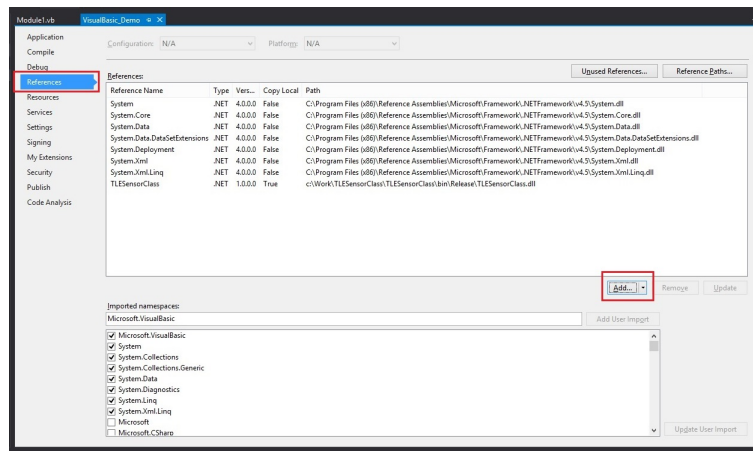


Figure 6: Project Properties Window

4. In the newly opened window press **Browse** in the bottom of the window, locate TLESensorClass.dll and press **OK**.

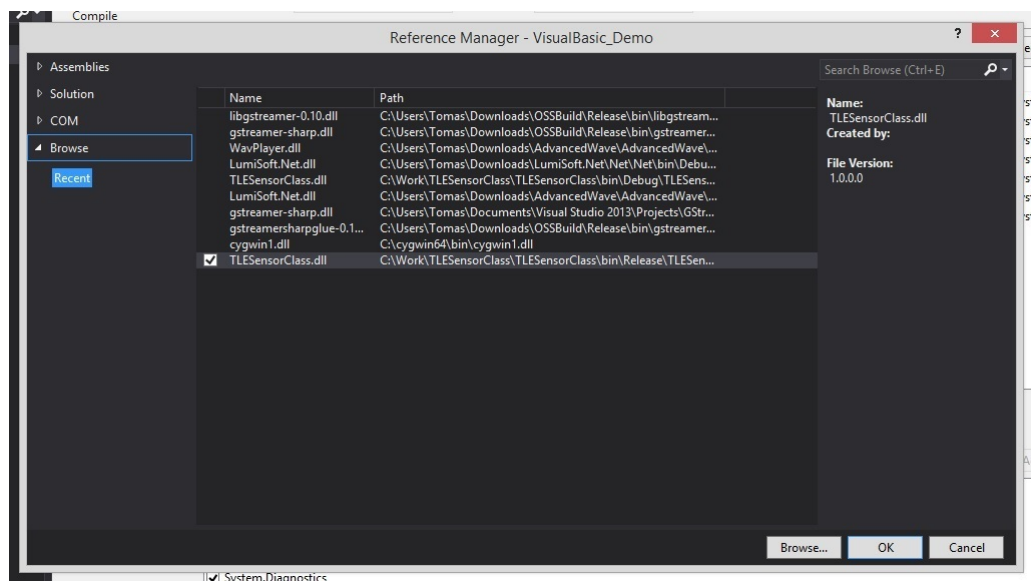


Figure 7: Project Properties Window

5. Now we need to set the target framework of the project to .NET Framework 4.5. In Solution Explorer right click on the project name and select **Properties**. In Properties menu set **Target framework** to **.NET Framework 4.5**.

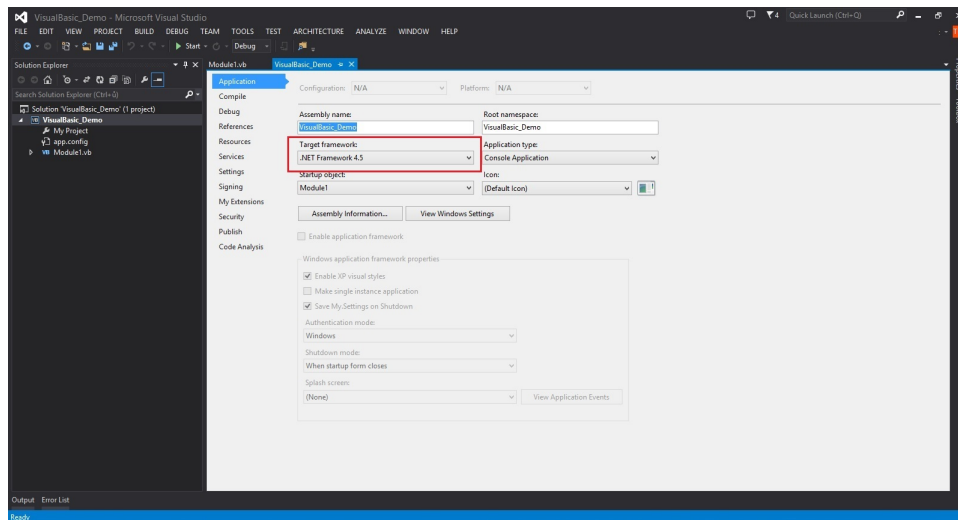


Figure 8: Project Properties Window

Application example

In this sample application we will just connect to sensor, read measurements and print them to console.

```
' Necessary import for using TLE Sensor classes
Imports TLESensorClass.Sensor
Imports TLESensorClass.Connections

Module Module1

    Sub Main()
        ' Definition of connection information – IP ↵
        address and port
        Dim con As New Connection_LAN("192.168.0.20", ↵
        1024)
        ' Creating sensor object with defined connection ↵
        data
        Dim sensor As New SensorClass(con)
        ' Connecting to sensor
        sensor.connect()
        ' Reading measurements and printing them to ↵
        console
        Console.WriteLine(sensor.get1Data())

    End Sub

End Module
```

2 List of TLESensorClass items

2.1 Connection Class

Constructors

Connection_LAN(IPAddress _ipAddress, int _port)

Connection_LAN(string _ipAddress, int _port)

Connection_LAN(string _ipAddress, string _port)

connects to the sensor using the ethernet interface and specified parameters

Parameters

int	BytesAvailable	Returns the number of bytes in the buffer
-----	----------------	---

Methods

bool	connect()	Connects to the sensor
bool	disconnect()	Disconnects from the sensor
Status	testConnection()	Tests the availability of the sensor and returns Online, Occpupied and Offline states
byte[]	ExecuteCommand(byte _command, byte[] _data, int _respondSize)	Sends the specified command to sensor and returns specified amount of data

2.2 EEPROMEthernet class

Constructors

EEPROMEthernet(byte[] _inputArray)

constructor with an input array as a the parameter

EEPROMEthernet(IPAddress _ip, IPAddress _subnet, IPAddress _gateway)

constructor with input parameters

Parameters

const int	DATA_SIZE	the lenght of the input array
const int	PAGE_NUM	the number of the page in EEPROM memory
IPAddress	IPAddress	the IP address of the sensor
IPAddress	IPSubnetMask	the Subnet mask
IPAddress	IPGateway	the Gateway

2.3 EEPROMParameters class

Constructors

EEPROMParameters(byte[] _array)

constructor with an input array as the parameter

EEPROMParameters(int _startLineROI, int _numOfLinesROI, int _averageWindow,
int _threshold, int _timeOfIntegration, int _laserPower, int _offsetY, int _offsetX,
int _defaultMode, int _lineProcMode, byte _peakSizeTo, byte _peakSizeFrom,
int _startColumnROI, int _numOfColumnsROI, int _startColumnWIN, int
_numOfColumnsWIN, int _startLineWIN, int _numOfLinesWIN)

constructor with an input parameters

Parameters

const int	DATA_SIZE	the lenght of the input array
int	AverageWindow	the Average Window
int	Threshold	the Threshold
int	TimeOfIntegration	the Time of Integration
int	LaserPower	the Laser Power
int	OffsetY	the Value of the height offset
int	OffsetX	the Value of the distance offset
int	DefaultMode	the Default measuring mode
int	LineProcMode	the Line Processing Mode
int	PeakSizeFrom	the Start value of the peak size
int	PeakSizeTo	the End value of the peak size
int	StartLine_limits	the Start line of ROI
int	NumberOfLines_limits	the Number of lines of the ROI
int	StartColumn_limits	the Start column of the ROI
int	NumberOfColumns_limits	the Number of columns of the ROI
int	StartLine_win	the Start line of the Window
int	NumberOfLines_win	the Number of lines of the Window
int	StartColumn_win	the Start column of the Window
int	NumberOfColumns_win	the Number of columns of the Window
byte[]	EEPROMParameters_field	the parameters in form of a byte array

2.4 PointSize class

Constructors

PointSize(int _pointX, int _pointY, int _sizeWidth, int _sizeHeight)
constructor with separated values as parameters

PointSize(Point _point, Size _size)
constructor with different types of parameters

Parameters

Size	Size_	the Value of the size
Point	Point_	the Value of the point
int	PointX	the X coordinate of the point
int	PointY	the Y coordinate of the point
int	Width	the Width
int	Height	the Height
int	Area	the Calculated area
int	EndY	the Y coordinate of the end point
int	EndX	the X coordinate of the end point

Methods

PointSize	doubleValues()	Doubles all values of the PointSize
PointSize	halfValues()	Divide values of the PointSize in half

2.5 ProfileData class

Constructors

ProfileData(int _length)

creates profile data instance with given length

Parameters

int[]	Distance	Integer array with distance values
int[]	Height	Integer array with height values

2.6 SensorDataFormat class

Constructors

SensorDataFormat(byte[] _data)

creates profile data instance with given length

Parameters

int	ValueX	the Distance value
int	ValueY	the Height value
byte	MODE	the Active measuring mode
bool	USER_PAR_CHG	user parameters has changed
bool	OVER410_CNT	
bool	ZERO_CNT	

2.7 SensorClass class

Constructors

`public SensorClass(Connection _connection)`

makes instance of the SensorClass with given connection data

Parameters

`int BytesAvailable` Returns the number of bytes in the buffer

General Methods

`public bool connect()`

Connects to the sensor using specified connection information.

`public bool disconnect()`

Disconnects from the sensor.

`public string getFWVersion()`

Reads the sensor firmware version as a string.

`public byte[] readData(int _respondSize)`

Reads specified amount of data from the sensor.

`public byte[] sendCommand(byte[] _commandData)`

Sends specified command to the sensor without specifying the respond size.

`public byte[] sendCommand(byte[] _commandData, int _respondSize)`

Sends specified command to the sensor and reads specified amount of data.

`public byte[] setMode(byte _modeNum)`

Sets specified measuring mode.

`public byte[] sensorUnlock2()`

Sends second level unlock command to the sensor.

`public byte[] startProfileStream()`

Starts profile data stream.

`public byte[] stopProfileStream()`

Stops profile data stream.

`public byte[] sensorUnlock()`

Sends unlock command to the sensor.

Methods for reading measured data

`public SensorDataFormat get1Data()`

Reads measurements and returns them as a SensorDataFormat instance.

`public byte[] getImageFull()`

Reads full resolution image.

`public byte[] getImageReduced()`

Reads quarter resolution image.

`public ProfileData getProfile()`

Reads profile measurements and returns them as a ProfileData instance.

`public byte[] getProfileBytes()`

Reads profile measurements and returns them as a byte array.

Methods for working with EEPROM memory

```
public EEPROMParameters getEEPROMParameters(byte _pageNum)
```

Reads parameters from the specified page of the EEPROM memory.

```
public byte[] setEEPROMParameters(EEPROMParameters _eepromData,  
byte _pageNum)
```

Sets data to the specified page of the EEPROM memory.

```
public EEPROMEthernet getEEPROMEthernet()
```

Reads the ethernet settings from the EEPROM memory.

```
public byte[] setEEPROMEthernet(EEPROMEthernet _eepromData)
```

Sets the ethernet settings to the EEPROM memory.

```
public byte[] setPage(byte _pageNum)
```

Activate the specified page of the EEPROM memory.

Methods for working with parameters

Methods for reading values return the value as an integer or byte. Methods for setting values returns byte array with the response from the sensor (usually echo command).

Parameters		
Average Window	GET	int getAverageWindow()
	SET	setAverageWindow(int _value)
Laser Power	GET	int getLaserPower()
	SET	setLaserPower(int _value)
Line Proc Mode	GET	int getLineProcMode()
	SET	setLineProcMode(int _value)
Peak Size From	GET	int getPeakSizeFrom()
	SET	setPeakSize(byte _from, byte _to)
Peak Size To	GET	int getPeakSizeTo()
	SET	setPeakSize(byte _from, byte _to)
Region of Interest	GET	PointSize getLimits()
	SET	setLimits(PointSize _value)
Threshold	GET	int getThreshold()
	SET	setThreshold(int _value)
Time of Integration	GET	getTimeOfIntegration()
	SET	setTimeOfIntegraion(int _value))
Window	GET	PointSize getWindow()
	SET	setWindow(PointSize _value)

3 TLESensorClass Examples

The TLESensorClass DLL consists of several classes. Some of them are used for formatting input data, some of them for connecting to the sensor and all the classes are encapsulated in one main class - **SensorClass**. In series of examples will be shown how to connect to sensor, read measured data, read image data and work with parameters and EEPROM memory.

3.1 Connecting to sensor

When connecting to the sensor, an instance of Connection class have to be made. This is important because the TLE1 sensor has two versions – one with the ethernet interface and one with the USB interface. If you want to connect to the ethernet interface, use this line:

```
Connection con = new Connection_LAN("10.226.164.156", ↵  
1024);
```

You have to specify the target IP address and port. The sensor uses two ports for connecting – port 1024 and port 1028. For the USB connection use this line:

```
Connection con = new Connection_USB();
```

A method called **connectToSensor()** will be created, it will be used throughout all examples in this document. This method will use the ethernet connection and will return an instance of SensorClass if connecting was successful. If not, method will return null.

```

static SensorClass connectToSensor()
{
    Connection con = new Connection_LAN("10.226.164.156", ←
        1024);
    SensorClass sensor = new SensorClass(con);
    if (sensor.connect()) { return sensor; }
    else return null;
}

```

3.2 Working with parameters

The TLE1 sensor uses many parameters which affect measuring. A list of parameters can be found in TLE1 User's Guide document. In the example below the value of Time of Integration parameter will be read. This value is 2 B long, but is returned as a single integer.

```

static void Main(string[] args)
{
    SensorClass sensor = connectToSensor();
    if (sensor != null)
    {
        int tint = sensor.getTimeOfIntegration();
        PointSize window = sensor.getWindow();
        Console.WriteLine(window.Height);
    }
}

```

For setting Time of Integration value, the method setTimeOfIntegration have to be used and integer value have to be added as a parameter.

NOTE: The sensor itself is not checking whether this value is valid or not. This has to be done by the programmer.


```

static void Main(string[] args)
{
    SensorClass sensor = connectToSensor();
    if (sensor != null)
    {
        int timeOfIntegration = 1024;
        sensor.setTimeOfIntegraion(timeOfIntegration);
    }
}

```

3.3 Working with measured data

The sensor provides three types of measured data – measured value, image data and profile data.

Measured value

The measured value consists of the distance value, height value and other parameters in the auxiliary byte. For covering this structure there is a class named **SensorDataFormat**. This class makes reading of measured values easier because the sensor returns all values in 5 B long byte field.

```

static void Main(string[] args)
{
    SensorClass sensor = connectToSensor();
    if (sensor != null)
    {
        SensorDataFormat data = sensor.get1Data();
        Console.WriteLine(data.ValueX);
        Console.WriteLine(data.ValueY);
        Console.WriteLine(data.MODE);
    }
}

```

Image Data

Image data are represented as a byte array. The sensor provides two types of image data – full resolution and quarter resolution. The maximum resolution for the full image is 1280 x 1024 pixels and for the quarter image it is 640 x 512 pixels.

```
static void Main(string[] args)
{
    SensorClass sensor = connectToSensor();
    if (sensor != null)
    {
        byte[] imageFull = sensor.getImageFull();
        byte[] imageReduced = sensor.getImageReduced();
    }
}
```

Profile Data

The profile data are preprocessed image data. The sensor sends this data as a byte array. To make this structure easier to read, the class **ProfileData** can be used. This class splits the input byte array into two integer arrays. To read this data as a byte array, the method `getProfileBytes()` has to be used.

```
static void Main(string[] args)
{
    SensorClass sensor = connectToSensor();
    if (sensor != null)
    {
        ProfileData data = sensor.getProfile();
        int[] arrayDistance = data.Distance;
        int[] arrayHeight = data.Height;
    }
}
```

3.4 Working with EEPROM memory

The sensor contains 8 pages with user parameters and 1 page for the ethernet settings data. It would be very complicated to handle each parameter separately. Therefore, if there is the need to change some data in the memory, all parameters have to be set in once. The structure of parameters in the EEPROM memory is defined in **EEPROMEthernet** and **EEPROMParameters** classes.

```
static void Main(string[] args)
{
    SensorClass sensor = connectToSensor();
    if (sensor != null)
    {
        // READING DATA
        EEPROMEthernet eData = sensor.getEEPROMEthernet()↵
        ;
        IPAddress ip = eData.IPAddress;

        // reading of page 247
        EEPROMParameters pData = sensor.↵
        getEEPROMParameters(247);
        int threshold = pData.Threshold;

        // SETTING DATA
        EEPROMEthernet newData = new EEPROMEthernet(
            IPAddress.Parse("192.168.0.16"),
            IPAddress.Parse("255.255.255.0"),
            IPAddress.Parse("192.168.0.1"));

        sensor.setEEPROMEthernet(newData);
    }
}
```